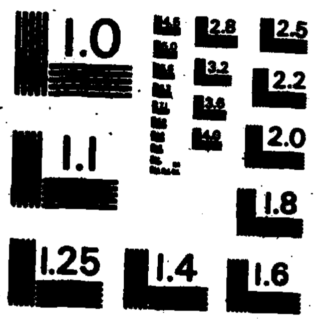
AD A135 257   ASYNCHRONOUS DISCRETE CONTROL OF CONTINUOUS PROCESSES   1/2
(U) NORTHEASTERN UNIV BOSTON MA M E KALISKI ET AL.
30 JUL 83 AFOSR-TR-83-0877 F49620-82-C-0080
UNCLASSIFIED   F/G 9/3   NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0877 — 2. GOVT ACCESSION NO. AD-A135257	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Asynchronous Discrete Control of Continuous Processes	5. TYPE OF REPORT & PERIOD COVERED Annual Report 7/1/82-6/30/83
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Martin E. Kaliski, Timothy L. Johnson	8. CONTRACT OR GRANT NUMBER(s) F49620-82-C-0080
9. PERFORMING ORGANIZATION NAME AND ADDRESS Northeastern University, 360 Huntington Avenue, Boston, MA 02115	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A3
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Math. & Info. Sciences, Air Force Office of Scientific Research, Bolling AFB, D.C. 20332	12. REPORT DATE 7/30/83
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

coding, automata theory, discrete control, switching theory, feedback control

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This research concerns the analysis and synthesis of asynchronous discrete-state or hybrid-state feedback compensators for continuous or hybrid-state processes. New realization theories for asynchronous discrete systems, based on automata and semigroup theory, have been derived. These theories suggest new architectures for asynchronous systems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ASYNCHRONOUS DISCRETE CONTROL OF CONTINUOUS PROCESSES

M.E. Kaliski and T.L. Johnson

July 1983

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Prepared by:

Northeastern University
360 Huntington Avenue
Boston, MA 02115

Prepared for:

Mathematical and Information Sciences Directorate
Attn: J. Burns
Air Force Office of Scientific Research
Bolling Air Force Base
Washington D.C. 20332

TABLE OF CONTENTS

1. INTRODUCTION AND STATEMENT OF WORK

This report summarizes the research undertaken during the period from July 1, 1982 through June 30, 1983, the first year of a three-year research program entitled "Asynchronous Discrete Control of Continuous Processes".

The research during this first contract year centered about developing sound theoretical models for asynchronous systems, models that extend and generalize previously developed research of the co-investigators on synchronous discrete control. The following sections of this report delineate this year's work.

The original proposal for this research described four tasks:

(1.1)	Analysis of Qualitative Properties of Asynchronous Hybrid Systems
(1.2)	Acceptance and Control for Asynchronous Hybrid Systems
(1.3)	Linguistic Approach to Asynchronous Systems
(1.4)	Application to Real-time Multi-tasking Systems; Stochastic Analysis

These tasks are to be pursued in parallel over a 3-year period, with the first two tasks receiving greatest emphasis during the first year. The research involves a collaboration between Prof. M.E. Kaliski of Northeastern University and Dr. T.L. Johnson of Bolt Beranek and Newman Inc. Dr. Johnson has taken responsibility for Tasks 1.1 and 1.4, while Prof. Kaliski has overseen Tasks 1.2 and 1.3.

2. RESEARCH PROGRESS

2.1 Analysis of Qualitative Properties of Asynchronous Hybrid Systems

In order to focus on key issues in the realization of asynchronous hybrid systems, the class of dynamic systems with finite input, output and state sets -- termed "simple asynchronous machines" (SAM), was treated during this year. The time-variable was assumed to be real-valued. Interesting results were obtained under the mild assumption that the input is a piecewise continuous (i.e., piecewise-constant) function. Under these conditions, the semigroup properties can be used to show that such systems can be realized in terms of a finite set of constant-input state-response functions that play a role analogous to the impulse response functions of linear system theory. Preliminary results reported in Johnson and Kaliski (1983) show that in the time-invariant case, such a realization can be synthesized from ideal digital components similar to those which are commercially available -- but in an architectural configuration which is new.

The key concept in deriving this new realization is that each input transition triggers a "cascade" of subsequent state transitions which is predetermined by the machine structure; this cascade may be overwritten by cascades due to subsequent input transitions, so that the output timing reflects a sequence of cascades keyed to input transition times. Various architectural configurations have been described for generating the fundamental state transition sequences; the applicable architecture depends on the specializing assumptions which are introduced.

This realization theory overcomes a number of fundamental problems associated with alternative representations. First, it places the theory of asynchronous discrete systems in a clear relationship with the theory of semigroups, which is almost universally used in describing a wide variety of other classes of dynamic systems: this link has been missing from previous results on asynchronous machines. Secondly, it successfully resolves a problem which occurs if transition rates are bounded by assumption -- i.e., that it is unreasonable to impose constraints on input transition times that depend on internal state dynamics. The ideal mathematical representation is well-defined even for unbounded or infinite transition rates; the transition rate bounds arise only when the differences between real and ideal synthesis components are considered. This is a key requirement in the consideration of general feedback systems composed of simple asynchronous machines, since a feedback system should itself be representable as a simple asynchronous machine! For instance, unstable feedback systems are well-defined and are realizable, so that a meaningful theory of stability can be developed.

As anticipated, this theory suggests the nature of solutions to a host of problems which can now be resolved in subsequent research: (1) the cases of hybrid state, input and output sets, respectively, can be considered; (2) the duality between the smoothness of the input-state and state-output mappings can be examined, and "well-behaved" asynchronous machines can be defined; (3) feedback system properties such as controllability, observability and stability can be defined for simple asynchronous machines; (4) equivalent realizations which partition the timing and data flow functions of asynchronous machines can be sought.

From a practical standpoint, the most significant implication of the realization results is the possibility that a general automatic synthesis procedure for asynchronous state machines can be developed. The theory suggests a general architecture which is significantly different from traditional Von Neumann and Data Flow architectures, which require considerable use of ad hoc design methods by experienced designers. A general automatic synthesis procedure could have major implications for computer-aided circuit design and achievable performance of VHSIC chips, as well as suggesting new concepts for the design of asynchronous feedback systems. A separate effort along these lines is anticipated.

2.2 Acceptance and Control for Asynchronous Hybrid Systems

Efforts during this first contract year focused on developing appropriate models and tools for characterizing asynchronous coders. These coders, as interfaces between the plant to be controlled, and the finite-state controller itself, are intrinsically hybrid devices which, as asynchronous devices, react not only to input changes but to the times at which these changes occur. Thus models of coders explicitly incorporating transition times were sought.

A mathematical abstraction of the asynchronous coding process can be made by viewing such coders as mappings from a subset of (RxR)+, the set of non-null, finite length sequences of points in RxR (R the real numbers) into the binary alphabet {0,1}. This abstraction, detailed in the attached informal memorandum "Towards a Theory of Finitary Asynchronous Coders", and summarized in the paper "Towards a Theory of Asynchronous, Real-Time Coders and Their Applications to Discrete-Control of Continuous Processes", 1983 American Control Conference, allows us to view asynchronous coders as special forms of their synchronous counterparts -- forms defined over a limited subset of allowable input strings. This subset consists of strings whose second components (namely time), always increases.

Having made this abstraction, research turned to finding meaningful ways to extend such "partially specified" maps to all of RxR so as to permit our previously developed theory of synchronous coders to come into play. The memorandum cited above develops such an extension theory for the case of finitary (finite-state) coders and introduces a key property of such coders -- tight structure. This property is a non-trivial one whose implications, as of this writing, are still being researched.

The tightly structured sequences that form the domain of asynchronous coders may be viewed as piecewise constant mappings on the real numbers, and thus it is only natural that attention turned to characterizing such mappings. The distribution of the "break points" of such mappings was the focus of this research -- the breakpoints, after all, represent event "switching times". We desired to develop more general characterizations of breakpoint distribution than the simple model introduced in the cited memorandum above.

Such a problem has an ergodic flavor to it and some time was devoted to extending some previously-unfunded thesis research by one of Prof. Kaliski's doctoral students (Y. Kwankam) to develop more precise notions of such concepts as "orbital behavior" and "chaotic distributions". This extension, submitted to the International Journal of Systems Science under the title "A Theory of Orbital Behavior in a Class of Nonlinear Systems: 'Chaos' and a Signature-Based Approach" (Kaliski, Kwankam, and [illegible]) makes explicit the concept of orbital signature -- a concept which is intrinsically related to switching times. (The functions considered are piecewise monotone with two pieces on the [illegible] [illegible] [illegible] "side" of the function can record [illegible] [illegible] [illegible] [illegible] [illegible] and switches to the [illegible] other side. The study of such timing functions is currently being done.)

2.3 Linguistic Approaches to Asynchronous System Modelling

The above theoretical models will be used in subsequent research to characterize wider classes of asynchronous coders. We will attempt to extend the linguistic models that were so successful in characterizing synchronous coders to the asynchronous case.

2.4 Application to real-time Multitasking Systems; Stochastic Analysis

Only preliminary investigations of this topic have been completed this year. The key issue of interest in multitasking systems is the development of a fundamental model for process synchronization; i.e. two concurrent processes may be represented as parallel asynchronous machines connected by a communication channel which may impose requirements for synchronization at certain stages of a computation. A dual view of this problem is to determine when a single process can be split into two concurrent processes with minimal communications requirements. The representation of the (asynchronous) communications channel is being viewed (in the deterministic case) from the perspective of the realization theory of asynchronous machines. As a preliminary example, the partial realization of a UART (universal asynchronous receiver-transmitter) has been worked out this year.

Stochastic problems have not been considered this year. However, it should be noted that the semigroup theory of stochastic systems is well-developed, and this provides a point of departure for developing a theory of stochastic asynchronous machines via the results cited in Section 2.1.

3. PUBLICATIONS

The following works have been published and/or submitted for publication this year.

Kaliski, M.E., "Towards a Theory of Finitary Asynchronous Coders", Northeastern University Memorandum, January 1983.

Kaliski, M.E., Kwankam, S.Y., and Halpern, P., "A Theory of Orbital Behavior in a Class of Nonlinear Systems: 'Chaos' and a Signature-based Approach", submitted to Intl. J. Systems Science, February 1983.

Kaliski, M.E. and Wimpey, D.G., "Towards a Theory of Asynchronous, Real-time Coders and Their Applications to Discrete Control of Continuous Processes", Proc. American Control Conference, San Francisco, CA, June 1983, pp. 731-733.

Wimpey, D.G. and Johnson, T.L., "Finite-State Control of Continuous-State Processes: The Discrete Time Case", Proc. 21st IEEE Conf. on Decision and Control, Orlando, FL, December 1982, pp. 1230-1232.

Johnson, T.L. and Kaliski, M.E., "Realization of Asynchronous Finite-State Machines", submitted to IEEE Trans. Auto. Contr. and 22nd IEEE Conf. on Decision and Control, San Antonio, TX, December 1983.

4. INTERACTIONS

Prof. Kaliski and Dr. Johnson met regularly every 1-2 weeks to coordinate research progress during the year. Prof. D.G Wimpey of Northeastern University has continued to participate as an unfunded collaborator in this line of research.

A proposal by BBN, Inc. to develop control system design algorithms based on our earlier results for the discrete-time case was submitted to Wright-Patterson AFB but not funded this year for lack of program funds. A similar effort has since been funded by the Army R&D Command.

Dr. Johnson has been involved in related research on modular hierarchical (hybrid-state) control systems and has given a number of presentations and lectures during the year on this topic.

FINITE-STATE CONTROL OF CONTINUOUS-STATE PROCESSES: THE DISCRETE TIME CASE*

David G. Wimpey
Department of Electrical Engineering
Room 409 Dana
Northeastern University
Boston, Mass. 02115

Timothy L. Johnson
Bolt, Beranek and Newman, Inc.
10 Moulton Street
Cambridge, Mass. 02238

and

Laboratory for Information and Decision Systems
Room 35-205B
Massachusetts Institute of Technology
Cambridge, Mass. 02139

ABSTRACT

An algorithm for the design of finite-state compensators for nonlinear discrete-time systems over R^n is developed. The problem is formulated as a regulator problem and the solution, if it exists, includes specification of all A/D and D/A quantization levels and is exact in the sense that the plant response is guaranteed. The solution procedure involves finite-state aggregation of the plant and the design of a controller for the resulting nondeterministic machine.

I. Introduction, Problem Formulation

The plant to be controlled is a finite-dimensional discrete-time system $\mathcal{P} = (R^n, R^m, R^p, f, g)$, with state-transition map $f\colon R^n \times R^m \to R^n$ and readout map $g\colon R^n \to R^p$. We formulate the control problem as a regulator problem: Informally, the plant state x is to be regulated from some initial state set $X_o \subseteq R^n$ to a target set $T \subset R^n$ in some given time frame. The constraint on the regulator, or compensator, is that it must be finite-state realizable. The compensator is thus the system $\mathcal{H} = (H, R^p, R^m, \xi, \eta)$ with H a finite set, $\xi\colon H \times R^p \to H$ and $\eta\colon H \times R^p \to R^m$.

Definition 1: Infinite Horizon Problem

The finite-state system $\mathcal{H}$ is a weak regulator for $\mathcal{P}$, a target set $T \subset R^n$ and an initial state set $X_o \subset R^n$ if there is a state $h \in H$ such that, for each $x_o \in X_o$, solving the closed-loop equations

$$x_{k+1} = f(x_k, \eta(h_k, g(x_k)))$$

$$h_{k+1} = \xi(h_k, g(x_k))$$

with $h_o = h$ results in $x_k \in T \ \forall k \geq N$, where N is some positive integer.

The statement of the finite planning horizon problem is similar except that we only require x_k to reach T within the planning horizon N. The term "weak" refers to the fact that the compensator initial state is fixed. $\mathcal{H}$ would be a strong regulator for $\mathcal{P}$ if h_o could be chosen arbitrarily. In general, a strong finite-state regulator for $\mathcal{P}$ does not exist with either planning horizon; further conditions on $\mathcal{P}$, X_o and T are required. In this paper, we consider only the design of a weak regulator over an infinite planning horizon.

The problem formulation and terminology used here is standard. The problem of definition 1 has been called the synchronous (meaning weak) regulator problem for the case where $\mathcal{P}$ is a deterministic finite automaton (Gatto et al. [illegible]) and the weak regulator problem (Sontag [2]) for piecewise linear systems. The problem formulation is also similar to the target tube reachability problem for stochastic systems with a set-theoretic description of the uncertainties (Sira Ramirez [3]).

In general, we will only require f and g to be piecewise continuous in all arguments. Our solution provides specification of the entire compensator, including all A/D and D/A quantization levels. Furthermore, if a solution is found, it is exact in that the closed loop performance is guaranteed. An approximate solution to a similar problem is given by Riordan [4].

The proofs to all Propositions and Theorems may be found in [5].

II. Plant Aggregation

The design of a finite-state compensator $\mathcal{H}$ for $\mathcal{P}$ is based on a finite-state aggregate model of $\mathcal{P}$. This model is obtained as follows: Let $P = \{P_1, \ldots, P_r\}$ be a finite partition on R^n defined by the equivalence

*This research was performed at the M.I.T. Laboratory for Information and Decision Systems with support provided by the Air Force Office of Scientific Research Directorate of Mathematical and Information Sciences (Contract F49620-80-C-0002). Views expressed in this paper do not necessarily represent those of the U.S. Government.

relation $\equiv$. Then the agregate model state-transition map δ_P: $P \times R^m \rightarrow 2^P$ is defined as

$$\delta_P(P_i, u) = \{P_j \in P: \ f(P_i, u) \cap P_j \neq \phi\} \text{ for } u \in R^m.$$

Thus the nondeterministic finite-state system (see [6]) $\mathcal{S} = (P, R^m, \delta_P)$ simulates $\mathcal{P}$ in the following sense: If the initial state of $\mathcal{S}$ is the equivalence class $[x_o]_\equiv$ i.e. the block containing x_o, and the input sequence w is applied to both $\mathcal{S}$ and $\mathcal{P}$, then the set of possible states reached by $\mathcal{S}$ contains the equivalence class (modulo $\equiv$) of the state reached by $\mathcal{P}$.

III. Existence of Weak Finite-State Compensators

Clearly, a necessary condition for the existence of a weak finite-state regulator for $\mathcal{P}$ is that the initial states in X_o be controllable to T. To examine this condition in terms of the aggregate model $\mathcal{S}$, we may compute the attainability sets [3]:

$$T^{-k} \triangleq \{p \in P: \exists u \in R^m \text{ s.t. } \delta_P(p,u) \subset T^{-k+1}\}$$

$$k = 1, 2, \ldots$$

where $T^o = T_P \triangleq \{p \in P: p \subset T\}$ is the aggregate target set. Then if X_o is covered by T^{-k} for some k, i.e. $X_o \subset (\cup[p: \ p \in T^{-k}])$, there exists a state-feedback law for $\mathcal{S}$ (and hence $\mathcal{P}$) which solves the finite horizon regulator problem. Since the states of $\mathcal{P}$ are not directly measurable in general, the selection of the control inputs to be applied at any time step must be based on an estimate of the state of $\mathcal{P}$. To obtain a finite-state state-estimator we will use aggregate state-estimates i.e. estimates of the state of $\mathcal{S}$.

In principle the, the design of $\mathcal{H}$ may be carried out by the construction of a tree of aggregate state-estimates of $\mathcal{P}$. The root of the tree is the aggregate initial state set $X_o \triangleq \{[x]_\equiv \in P: \ x \in X_o\}$ and the nodes are elements of 2^P. If P_{k+1} is a successor node to node P_k, with the connecting arc labelled (u,y), then P_{k+1} is the new aggregate state estimate of $\mathcal{P}$ given that the plant output was y and then input u was applied. Of course, the arcs would have to be labelled by subsets, instead of elements, of $R^m \times R^p$, since there are only finitely many distinct state-estimates. The design is completed by selecting a feedback law K: $2^P \times R \rightarrow R^m$ which results in all possible paths of the tree having successive nodes which are eventually always subsets of T_P.

The design of an (open-loop) finite-state state-estimator for $\mathcal{P}$ proceeds as follows. Define an equivalence relation $\approx$ on R^p as $y_1 \approx y_2 \Leftrightarrow \{[x]_\equiv \in P: g(x) = y_1\} = \{[x]_\equiv \in P: g(x) = y_2\}$. Thus, $y_1 \approx y_2$ iff the set of aggregate states that $\mathcal{P}$ may be in given output y_1 is identical to the set given y_2. Now define q: $R^p \rightarrow R^p/_\approx$ ($\triangleq W$) and G: $W \rightarrow 2^P$: $w \mapsto \{[x]_\equiv: g(x) \in w\}$. Then the state-estimator is $\mathcal{S} = (2^P, R^m \times R^p, \alpha)$, where α: $2^P \times R^m \times R^p \rightarrow 2^P$ is defined in terms of $\bar{\alpha}$: $2^P \times R^m \times W \rightarrow 2^P$ as

$$\alpha(\hat{x}, u, y) = \bar{\alpha}(\hat{x}, u, q(y))$$

with $\quad \bar{\alpha}(\hat{x}, u, w) = \cup[\delta_p(p,u): p \in \hat{x} \cap G(w)]$.

Proposition 1: The solution of

$$x_{k+1} = f(x_k, u_k)$$
$$\hat{x}_{k+1} = \bar{\alpha}(\hat{x}_k, u_k, q(g(x_k)))$$

with $\hat{x}_o = [x_o]_\equiv$ results in $x_k \in [x_k]_\equiv \in \hat{x}_k$ for all $k \geq 0$.

Note that the quantization map q depends only on g and P. The map $\bar{\alpha}$ shows explicitly the decomposition of α . Also, in order to compute δ_p and q we require that f and g be piecewise continuous.

Proposition 2: A weak finite-state regulator $\mathcal{H}$ exists for $\mathcal{P}$, T and X_o if there exists a map K: $2^P \times W \rightarrow R^m$ with the property that solutions of the system

$$x_{k+1} = f(x_k, K(\hat{x}_k, q(g(x_k))))$$
$$\hat{x}_{k+1} = \bar{\alpha}(x_k, K(\hat{x}_k, q(g(x_k))), q(g(x_k)))$$

for any $x_o \in X_o$, $\hat{x}_o = X_{oP}$ result in $\hat{x}_k \in T_P \ \forall k \geq N$.

$\mathcal{H}$ then has $H = 2^P$ and

$$\xi(h,y) = \bar{\alpha}(h, K(h, q(y)), q(y))$$

$$\eta(h,y) = K(h, q(y)).$$

IV. Solution of the Infinite Horizon Problem

In practice, construction of the successor tree for $\bar{\alpha}$ from the root down is highly inefficient. A "bottom-up" procedure is desirable, but this requires knowledge of a set of state-estimates in T_P for which all successive state-estimates will remain in T_P. Essentially, this involves the computation of a steady-state reduced target tube [3] which we denote T_P^R .

Definition 2: Define T_P^R to be the largest subset of 2^{T_P} with the property that $P' \in T_P^R \Leftrightarrow \forall w \in W$ s.t. $G(\omega) \cap P' \neq \phi$, there exists $u \in R^m$ s.t. $\bar{\alpha}(P', u, w) \in T_P^R$.

An algorithm for the computation of T_P^R is given in [5]. This algorithm actually computes a minimal cover of T_P^R, namely $\bar{T}_P$; thus if $P' \in \bar{T}_P$ then all non-empty subsets of P' are not included in $\bar{T}_P$.

Once $\bar{T}_P$ has been computed, the tree for $\bar{\alpha}$ may be constructed as follows:

$$X^o = \bar{T}_P$$
$$X^{i+1} = \{P' \in 2^P: \text{ for each } w \in W \text{ s.t. } G(w) \cap P' \neq \phi, \text{ there exists } u \in R^m \text{ s.t. } \bar{\alpha}(P', u, w) \in X^i\}.$$

Theorem: A weak finite-state regulator $\mathcal{H}$ exists for $\mathcal{P}$, T, and X_o if there exists a finite partition P of R^n such that $X_{OP} \subset X'$ for some $X' \in X^N$ and some non-negative integer N.

The X^i are essentially attainability sets constructed from T_P. A detailed algorithm for the computation of the X^i may be found in [5]. The tree specifying $\bar{\alpha}$, when represented as a graph, is the state-transition graph for a system with input set $R^m \times W$ i.e. each arc is labelled as (V, w) for some $V \subset R^m$, $w \in W$. Thus, corresponding to each $P' \in X^{i+1}$ and each w such that $G(w) \cap P' \neq \phi$, we have a transition set V; then if $u \in V$, $\bar{\alpha}(P', u, w) \in X^i$. Since there are only finitely many sets V we may select one representative $\bar{u}$ from each and define K as $K(P', w) = \bar{u}$, i.e. K has finite range $\bar{V} \subset R^m$. The structure of *H* is shown in Figure 1.

Clearly, the existence of the regulator *H* (if it exists at all!) depends on the choice of P. Initial choices for P should be coarse to simplify the controller, and refinements should be made if *H* cannot be found. The design process is therefore iterative. Techniques for choosing the initial partition and for updating or refining P are under investigation. These are complex problems which are problem dependent. To simplify the design, a heirarchical controller structure has been suggested [5].

V. Discussion, Conclusions

We have described a new approach to the design of dynamic digital compensators. These compensators are finite-state and are therefore directly realizable using digital logic circuitry. Our procedure specifies all quantization levels and takes all quantization effects into account.

Previous attempts to design finite-state controllers for continuous-state systems were concerned with memoryless switching controllers. These problems were often formulated as optimal control problems and the design often employed approximations which resulted in closed-loop systems with performances that were difficult (if not impossible) to predict [4, 7].

The problem of quantization (or finite-state aggregation) of continuous-state dynamic systems is not new [8, 9]. Our approach follows most closely the ideas of Kornoushenko [10] and is related to the problem of obtaining structure-preserving covers for finite automata [6].

References

[1] Gatto, M., and Guardabassi, G., "The Regulator Theory for Finite Automata", Inf. and Control, Vol. 31, pp. 1-16, 1976.

[2] Sontag, E.D., "Nonlinear Regulation: The Piecewise Linear Approach", IEEE Trans., AC-26, No. 2, April 1981.

[3] Ramirez, S., "Set Theoretic Control of Large Scale Uncertain Systems", M.I.T., Ph.D. Thesis, Dept. of Elec. Eng. and Comp. Sci., May 1977.

[4] Riordan, J.S., "Optimal Feedback Characteristics from Stochastic Automaton Models", IEEE Trans. AC-14, No. 1, February 1969.

[5] Wimpey, D.G., "Finite-State Control of Discrete-Time Continuous Processes: An Automata Motivated Approach", M.I.T., Ph.D. Thesis, Dept. of Elec. Eng. and Comp. Sci., January 1982.

[6] Hennie, F.C., Finite-State Models for Logical Machines, Wiley, 1968.

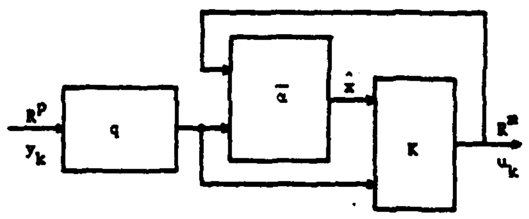
[7] Tou, J.T., Optimum Design of Digital Control Systems, Academic Press, New York, 1963.

[8] Wang, P.K.C., "A Method of Approximating Dynamical Processes by Finite-State Systems", Int. J. Control Vol. 8, No. 3, pp. 285-296, 1968.

[9] Borkar, V. and Varaiya, P., "Finite Chain Approx. for a Continuous Stochastic Control Problem", IEEE Trans., AC-26, No. 2, April 1981.

[10] Kornoushenko, E.K., "Finite Automaton Approximation of Behaviour of Linear Stationary Continuous Plants", Aut. and Rem. Control, No. 7, pp. 1092-1102, 1977.

Figure 1. Structure of Regulator *H*.



Realization of Asynchronous Finite-State Machines[*]

T.L. Johnson[**]
BBN, Inc.
10 Moulton Street
Cambridge, MA 02238

Ph. 617/497-3413

M.E. Kaliski
Northeastern University,
405 Dana Hall
360 Huntington Avenue
Boston, MA 02115

Ph. 617/437-3034

Abstract

Asynchronous finite-state machines accurately represent the action of computer control systems on continuous and discrete processes. Despite their importance, the lack of a general analytic representation for asynchronous machines has prevented the development of adequate techniques for analysis and design of finite-state asynchronous control systems, while ad hoc real-time control software has proliferated. In this paper, a general realization method is presented and is applied to a portion of a universal asynchronous receiver/transmitter (UART).

[*]Research supported by AFOSR Mathematical and Information Sciences Directorate under Contract F49620-82-C-0080.

[**] Preferred address for correspondence and return of proofs.

I. Introduction

A simple real-time control program (e.g., one which repeatedly polls a number of input lines, performs certain logical tests and computations, and then deposits results in several output registers) operates asynchronously. While such programs undoubtedly characterize most computer control applications in operation today, very little theory for the rigorous analysis (let alone design!) of such systems is currently available. In fact, technical standards for defining the behavior of such systems, short of multichannel timing diagrams, do not exist. Some consequences of this situation are as follows: (1) systems are designed by trial and error; (2) control software is not generally transferable from one application to the next; (3) prior performance estimates of proposed systems cannot be obtained; (4) system functions cannot be efficiently documented, short of providing complete hardware/software descriptions. Clearly, these consequences are undesirable.

A number of strategies for avoiding these problems have evolved, but none are very adequate. The practitioner, if he is even aware of the preceding problems, is generally quite content with the status quo. Trial-and-error debugging of software is fast and relatively inexpensive compared to analytical design procedures. For simple systems, this approach indeed works well; but for large systems (e.g., the U.S. Space Shuttle [1]), the complexity becomes overwhelming, and even exhaustive simulation can fail to reveal subtle timing errors. The response of the traditional feedback control designer, by contrast, is to force the system to operate synchronously (or with multirate sampling), and to reduce logical operations (as distinct from real-number operations) to an absolute minimum. While this approach is usually more amenable to analysis and documentation, it forces the designer to overlook certain attractive control strategies and forces the processor to operate inefficiently; thus, performance may be significantly compromised [2].

The realization problem for asynchronous finite-state machines is a first step in the development of analytical methods for the design of asynchronous computer control systems. <u>The primary issues of interest are to define a natural set of primitives that can be used to characterize the behavior of asynchronous discrete-state systems, and to illustrate a general synthesis procedure whereby specification of the primitives leads to a realistic implementation</u>. In previous research of the authors and a colleague [3], these objectives have been substantially achieved for discrete <u>synchronous</u> systems. In <u>asynchronous</u> systems, the actual timing of events may effect the evolution of states, whereas in synchronous systems, only the chronological ordering of events is important (i.e., "synchronous" does not necessarily imply a uniform sampling rate).

Several simplifications are made here in order to isolate the main issues: (1) only the purely finite-state case is considered, although the technical framework admits an extension to hybrid-state systems; (2) feedback interconnection of systems is not considered, though with certain additional conditions, the issue of closedness under feedback could be addressed; (3) existence of invariants, equivalent representations, and minimality are not treated here. While the present results are thus incomplete, they do focus on pragmatic issues of interest.

II. Background and Notation

Conceptual models for asynchronous state machines have been employed primarily in automata theory, switching and communications theory, and digital circuit design. Kohavi [4] provides a reasonable overview of the available methods and technical issues, while Unger [5] Miller [6] have given detailed design examples for such systems. Petri nets [7] have also been used to represent the behavior of asynchronous machines, though they are often impractical for large systems. Kalman, Falb and Arbib [8] have developed a conceptual framework for mathematical systems theory which includes both automata theory and dynamic systems theory, but this is based primarily on formal analogies between existing theories; they did not explore the specific properties of hybrid systems, and in particular discrete-state, continuous time systems. In [9], an axiomatic framework for the representation of hybrid-state systems was introduced, and the special case of asynchronous machines was briefly discussed.

The key technical issue in developing a representation for asynchronous machines, aside from the choice of state variables, is essentially one of existence of solutions. Two aspects of this problem are: (1) On multiple input lines, transitions can occur at arbitrarily close times, which may lead to indeterminacy in the state transition function. (2) Even with constant inputs, the state may switch at a rate which increases so rapidly with time that its value is not continuable with respect to time. Both of these phenomena suggest the need to limit the maximum switching rate, or to impose a minimum switching delay time on the system. The avoidance of hazards and races is also an important practical design consideration for switching circuits that is closely related to these mathematical difficulties. If an asynchronous circuit is not designed to avoid these problems, its actual behavior may be indeterminate in the sense that successive trials with "identical" inputs produce different results, i.e., the behavior depends on details of design and implementation which are impractical to model. Limit cycle oscillations, one manifestation of the second problem above, may also occur in actual circuits, and their detailed structure is again implementation-dependent. These phenomena are almost always undesirable in practice; however, they are inherent in the discontinuous nature of the systems under study. In the next section, a further discussion is given of the way in which these phenomena are represented.

The general setting for these results is dynamic systems theory [10]. Only a slight generalization of the usual definition of a dynamic system is required for the current problem, due to the fact that the input and output sets admit only a discrete topology [8, pp. 163-164]. The general notation is introduced here, and is specialized in the following section.

Definition 2.1: A continuous-time dynamic system, Σ , on an open interval $T \subset R$, consists of

U an input set,
$\mathcal{U}$ an input space of functions $u: T \to U$,
Y an output set,
$\mathcal{Y}$ an output space of functions $y: T \to Y$,
X a state set,

a state transition map ϕ: $T \times T \times \mathcal{U} \times X \to X$, and a readout map $r: T \times T \times U \times X \to Y$. The state transition map is assumed to satisfy the following semigroup axioms

(i) Identity. $\phi(t,t,u(.),x_0)=x_0$ for all $t \in T$, $u(.) \in \mathcal{U}$, $x_0 \in X$.

(ii) Causality. For any elements $t_0, t \in T$, $t_0 \le t$, and any $x_0 \in X$, if $u_1(\tau) \equiv u_2(\tau)$, $t_0 \le \tau \le t$, then $\phi(t,t_0,u_1(.),x_0)=\phi(t,t_0,u_2(.),x_0)$.

(iii) Transitivity. For any $t_2 \geq t_1 \geq t_o$, all elements of T, any $u(.) \in \mathcal{U}$, and any $x_o \in X$, $\phi(t_2,t_o,u(.),x_o) = \phi(t_2,t_1,u(.),\phi(t_1,t_o,u(.),x_o))$

The dynamic system $\Sigma = (U,\mathcal{U},Y,\mathcal{Y},X,\phi,r)$ is thought of as the pair of equations

$$x(t) = \phi(t,t_o,u(.),x(t_o))$$

$$y(t) = r(t,t_o,u(t),x(t))$$

for $t \geq t_o$, both in T.

III. Main Results

In this section, the properties of asynchronous machines, as a subclass of dynamic systems, are examined, and corresponding ideal circuit realization results are given. Some preliminary notions are the following: notation follows that of the previous section.

Definition 3.1: A partial function of finite range (PFOF) is a partial function $z: T_o \to Z$, where $T_o \subseteq T \subseteq R$, and $Z = \{z^1, z^2, \ldots, z^q\}, q \geq 1$ is a finite set. A function of finite range (FOFR) is a PFOFR such that $\overline{T_o} = T$. (Overbar denotes closure with respect to the set operations on R).

Definition 3.2: A point $t \in T$ is a point of discontinuity of a FOFR $z: T \to Z$, with $q \geq 2$, if z takes on multiple values in every open set in T containing t. (Note: For $q=1$, points of discontinuity cannot exist).

Definition 3.3: A FOFR $z: T \to Z$, is a measurable function of finite range (MFOFR) if and only if the sets $T^i = \{t \in T \mid z(t) = z^i\}$ are measurable for $i=1,\ldots,q$, with respect to the Lebesque measure on R.

<u>Remark 3.1</u>: A FOFR $z: T \to Z$ with a countable number of points of discontinuity in every finite subinterval of T is measurable. This follows from properties of Lebesque-measurable sets.

<u>Definition 3.4</u>: A FOFR $z: T \to Z$ with a finite number of points of discontinuity in every finite subinterval of T is termed a <u>piecewise continuous function of finite range</u> (PCFOFR).

<u>Remark 3.2</u>: If z is a PCFOFR, it is measurable. In summary, PCFOFR $\subset$ MFOFR $\subset$ FOFR $\subset$ PFOFR (algebraic inclusion).

The special class of dynamic systems of interest here is the class of simple asynchronous machines.

<u>Definition 3.5</u>: A <u>simple asynchronous machine</u> (SAM) is a dynamic system $\Sigma = (U, \mathcal{U}, Y, \mathcal{Y}, X, \phi, r)$ with

$U = \{u^1, \ldots, u^m\}$, $m \geq 1$ a finite integer
$\mathcal{U} = \{u: T \to U \mid u \in \text{PCFOFR}, u \text{ is right-continuous}\}$
$Y = \{y^1, \ldots, y^p\}$, $p \geq 1$ a finite integer
$\mathcal{Y} = \{y: T \to Y \mid y \in \text{PCFOFR}, y \text{ is right-continuous}\}$
$X = \{x^1, \ldots, x^n\}$, $n \geq 1$ a finite integer

<u>Remark 3.3</u>: Functions in $\mathcal{U}$ and $\mathcal{Y}$ are required to be right-continuous in order that their values are well-defined for all $t \in T$, as required by Definition 2.1.

<u>Remark 3.4</u>: The case where X is not finite is also of interest, and will lead to more complex asynchronous machines, but these are not considered here.

These formal definitions are of interest only to the extent that the functions ϕ and r can be simplified, which is next shown to be the case. This is analogous to the expression of the transition mapping in terms of the transition matrix, for linear systems, and the expression of this in turn as a matrix of impulse-response functions that can be written in terms of a finite set of parameters for linear time-invariant systems.

A SAM can be thought of as a system

$$x(t) = \phi(t,t_o,u(\cdot),x_o)\varepsilon X \quad ; \ t\geq t_o \tag{3.1}$$

$$y(t) = r(t,t_o,u(t),x(t))\varepsilon Y; \ t,t_o\varepsilon T \tag{3.2}$$

Since $u\varepsilon$ PCFOFR and $[t,t_o]$ is a finite interval, u has a finite set of points of discontinuity, denoted $t_1,\ldots,t_N$, such that $t_o<t_1<t_2\ldots<t_N<t$ and

$$u(\tau) \equiv \begin{cases} u_o\varepsilon U & t_o\leq \tau < t_1 \\ u_i\varepsilon U, & t_i\leq \tau < t_{i+1},\ i\varepsilon\{1,N\}\text{(by right continuity)} \\ u_N\varepsilon U, & t_N\leq \tau < t \end{cases} \tag{3.3}$$

and furthermore, $u_{i+1}\neq u_i$.* [The degenerate cases m=1 and/or N=$\emptyset$ are not written out explicitly here]. By the causality property, ϕ in (3.1) depends only on u(.) in the interval $[t,t_o]$. By the semigroup property of ϕ,

$$\begin{aligned} x(t) &= \phi(t,t_N,u_N,\phi(t_N,t_{N-1},u_{N-1},\phi(\ldots,\phi(t_1,t_o,u_o,x_o)\ldots) \\ y(t) &= r(t,t_o,u_N,x(t)) \end{aligned} \tag{3.4}$$

*The case of a single input line with multiple values is considered here. For multiple lines, it is necessary to distinguish which input line has changed.

where (by slight abuse of notation), $u_o \ldots u_N$ represent constant functions of values $u_o \ldots u_N$, respectively, on the intervals in question.

From (3.4), it is apparent that the <u>fundamental solution</u>

$$\bar{\phi}(t,\tau,u,x): T \times T \times U \times X \to X;\ t \geq \tau \geq t_o; t,\tau \in T$$

is sufficient to define the transition mapping. This function is defined for a <u>finite</u> <u>set</u> of possible input-state combinations, and hence admits a sum-of-products (mini sum) decomposition in the form

$$\bar{\phi}(t,\tau,u,x) = \bigvee_{i,j} [\phi_{ij}(t,\tau)\ (u \Lambda u^i) \Lambda (x \Lambda x^j)] \qquad i=1,\ldots,m;\quad j=1,\ldots n \tag{3.5}$$

where V is the "or" operator and Λ is the "and" operator.

Furthermore, the range of ϕ_{ij} is finite for each i,j, and hence this function may be written as

$$\phi_{ij}(t,\tau) = \bigvee_{k=1}^{n} x^k\ \psi(t,T_{ij}^k(\tau)) \tag{3.6}$$

where $T_{ij}^k(\tau) = \{\sigma \geq \tau, \sigma \in T | \phi_{ij}(\sigma,\tau) = x^k\}$, and ψ is the characteristic function defined as

$$\psi(t,T^k) = \begin{cases} 1 & t \in T^k \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, by definition the sets $T_{ij}^k(\tau)$ have the completeness and orthogonality properties that for each $\tau \geq t_o$

$$\bigcup_{k=1}^{n} T_{ij}^k(\tau) = \{\sigma \geq \tau, \sigma \in T\} = T-(t_o,\tau)$$

$$T_{ij}^k(\tau) \cap T_{ij}^\ell(\tau) = \phi \quad \text{(the empty set)}$$

The readout map may be similarly specified:

$$r(t,\tau_o,u,x) = \bigvee_{ij} [r_{ij}(t,\tau)\ (u \Lambda u^i) \Lambda (x \Lambda x^j)] \quad i=1\ldots m,\ j=1,\ldots n \tag{3.7}$$

$$r_{ij}(t,\tau) = \bigvee_{k=1}^{p} y^k \, \psi(t, S_{ij}^k(\tau)) \; ; \; S_{ij}^k(\tau) = \{\sigma \geq \tau, \; \sigma \varepsilon T | r_{ij}(t,\tau) = y^k\} \tag{3.8}$$

Summarizing these results for time-varying SAM's, one obtains

<u>Theorem 3.1</u>: Any SAM is completely characterized by complete mutually orthogonal subsets $T_{ij}^k(\tau)$ and $S_{ij}^\ell(\tau)$, parameterized by τ and defined on $T-(t_o,\tau)$ where t_o is the infimal element of T. These sets have the following interpretation: $T_{ij}^k(\tau)$ is the set of times in T that the state takes on value x^k when the system starts in state x^j at time τ and the constant input $u=u^i$ is applied. $S_{ij}^\ell(\tau)$ is the set of times in T that the output takes on value y^ℓ when the system starts at time τ and the final state and input have values x^j and u^i.

In general, in the time-invariant case, it is well-known that $\phi(t,\tau,u(.),x)$ depends only on the difference $t-\tau$, and that $r(t,\tau,u(t),x(t))$ depends only on $u(t)$ and $x(t)$. The implications of the time-invariant assumption for the finite-state case are clear from the preceding constructions.

<u>Corollary 3.1</u>: Any time-invariant SAM is completely characterized by the complete mutually orthogonal subsets T_{ij}^k, of T; and the indicator function S_{ij}^ℓ. These sets have the following interpretation: T_{ij}^k is the set of times in T, relative to an initial time $t_o \varepsilon T$ that the state takes on value x^k when the system starts in state x^j at time t_o and the constant input $u \equiv u^i$ is applied. S_{ij}^ℓ takes value 1 if the current output value y^ℓ occurs whenever the current state takes value x^j and the current input takes value u^i, and is zero otherwise.

At this juncture, it should be noted that the assumptions about the input space allowed ϕ (and r) to be defined only on the set of all finite partitions of TxT (e.g., all possible "nets" that cover the plane); but this is all that is required. Secondly, it should be noted that the definition of a dynamic system does not specify any space of "state-functions". In fact, the sets T^k_{ij} in Corollary 3 may be almost arbitrarily ill-behaved. This means, in turn, that the rate of switching between different state values may actually be infinite. This situation exists because no continuity conditions have been imposed on ϕ or r. However, the assumption that the output be piecewise continuous does imply that certain compatibility conditions exist between the state transition and readout mappings.

<u>Corollary 3.2</u>: When a SAM is characterized as in Theorem 3.1 or Corollary 3.1, certain compatibility conditions among the sets T^k_{ij} and S^k_{ij} must exist in order to assure that the input-output mapping $\mathcal{W}_{t_o x_o}: \mathcal{U} \to \mathcal{Y}$ defined by

$$r(t,t_o,u(t),x(t)) = r(t,t_o,u(t),\phi(t,t_o,u(.),x_o))$$

takes $\mathcal{U}$ into (piecewise continuous functions) $\mathcal{Y}$ for all $x_o \epsilon X$. A necessary condition (in addition to completeness and orthogonality) for satisfaction of these conditions is that the sets $T^k_{ij}(\tau)$ and $S^k_{ij}(\tau)$ have a finite number of boundary points in every finite subinterval of $T-[t_o,\tau]$ for each $\tau \epsilon T$, in Theorem 3.1; or that the sets T^k_{ij} have a finite number of boundary points in every finite subinterval of T, in Corollary 3.1.

The necessary conditions of the Corollary can be established by noting that they imply that the state function x(t) is piecewise continuous in time and that the readout map preserves piecewise-continuity (in the time-varying case). These conditions are far from sufficient, though: the readout map may "mask" sequences of states which occur at infinite rates in the general case, so that "irregularity" in the state-transition function is compensated by "regularity" in the readout map.

For Corollary 3.1, it is apparent that the most interesting realization results will be obtained in the time-invariant case, and hence only time-invariant SAMs will be considered in the sequel. A useful insight is provided by (3.1): each input transition triggers a pattern of state transitions (depending on what "state" the system is in when it occurs) which persists only until the next input transition. The output thus may be viewed as resulting from a sequence of truncated cascades, each truncated cascade being drawn from a finite (but possibly large) set of waveforms. The definition of the output space for a SAM is such that the output can only make a transition whenever either the "state" or input makes a transition.

In considering circuit realization and synthesis issues, it is desirable to consider the subset of SAMs which have piecewise-continuous state functions.

Definition 3.6: A SAM is termed regular if for all $t_o \varepsilon T$, $u(.) \varepsilon \mathcal{U}$ and $x_o \varepsilon X$, $x(t)$ as defined by (3.1) is a PCFOFR when viewed as a function from T to X.

Necessary conditions for a SAM to be regular have been stated in Corollary 3.2. The ideal circuit realization for regular SAMs requires that certain idealized elementary building-blocks be defined.

Definition 3.7: A digital multiplexor (DMUX) is a memoryless element with address lines, $a_1 \ldots a_q$, where input a_i can assume n_i discrete values $\{a_i{}^j, j=1,\ldots,n_i\}$, $n_q = \prod_{i=1}^{q} n_i$ digital input lines labelled $v_{a_1 \ldots a_q}$ each taking n_o values, and one output line, w, taking n_o values, defined such that

$$w(t) = v_{a_1(t)a_2(t)\ldots a_q(t)}(t) \tag{3.9}$$

<u>**Remark 3.5**</u>: An (ideal) digital multiplexor can be synthesized from (ideal) digital switches.

<u>**Definition 3.8**</u>: A <u>digital function generator</u> (DFG) for the function $f:T\rightarrow Z$, where $T\subseteq R$ is an open interval and $Z=\{z^1\ldots z^q\}$ is a finite set, has a binary-valued input line, b, and a q-valued output line, z, defined such that

$$z(t) = f(t-\tau)$$

where τ is the time of the most recent $\emptyset\rightarrow 1$ transition on b.

<u>**Remark 3.6**</u>: Formally, the set of all discontinuity points of b(t) is partitioned into times $\{\tau_{\emptyset 1}\}$ and $\{\tau_{1\emptyset}\}$, and the supremum of $\{\tau_{\emptyset 1}\}\cap(-\infty,t)$ is defined to be τ.

<u>**Definition 3.9**</u>: An <u>ideal trigger element</u> $(IT)_{i\ell}$ <u>for the transition $i\rightarrow\ell$</u> on the input line v taking values $v^1\ldots v^q$ has a binary output b(t) such that

$$b(t) = \begin{cases} 1 & \text{for all } t \text{ such that } v(t^-) = \lim_{\tau\uparrow t} v(\tau) = v^i \text{ and} \\ & v(t^+) = \lim_{\tau\downarrow t} v(\tau) = v^\ell \\ 0 & \text{otherwise} \end{cases}$$

An <u>ideal transition element</u> (IT) with input line v detects all discontinuity points of v (i.e., its output is the union of all ideal trigger elements for that input line).

<u>**Remark 3.7**</u>: An ideal transition element is defined for all $v\varepsilon$POFR, while an ideal trigger element may be only defined for $v\varepsilon$PCPOFR. A one-shot circuit approximates an ideal $\emptyset\rightarrow 1$ transition element with a binary input line.

Definition 3.10: A logical transformation (LT) for the memoryless transformation G: $V \to Z$ where $V=\{v^1 \ldots v^s\}$ and $Z=\{z^1 \ldots z^q\}$ are finite sets with input line v and output line z, is defined such that

$$z(t) = Gv(t)$$

for (almost) all $t \in T$.

Symbols for these elements are shown in Figure 1. From the discussion preceding Theorem 3.1 and the remarks prior to Corollary 3.2, it is apparent that a SAM may be realized as shown in Figure 2. Every input transition initializes the function generators for $\phi_{ij}(\cdot,0)$ as defined in (3.5), and the multiplexor performs the logical operations $(u \wedge u^i) \wedge (x \wedge x^j)$ to select the appropriate function. The transformation S in Figure 2 is defined as in Corollary 3.1, based on the more general expression (3.8):

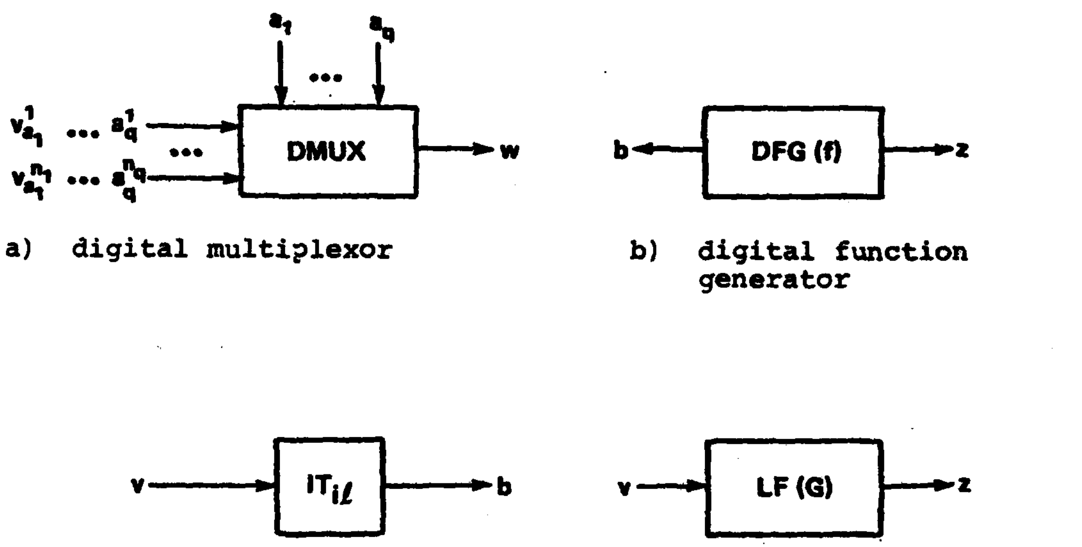
$$y = \bigvee_{i=1}^{m} \bigvee_{j=1}^{n} \bigvee_{\ell=1}^{p} y^{\ell} s_{ij}^{\ell} (u \wedge u^i) \wedge (x \wedge x_j) \qquad (3.10)$$

The interconnection of elements in Figure 2 has the effect of computing the transition mapping using the semigroup property according to (3.1). These results are restated as follows.

Theorem 3.2: A time-invariant SAM satisfying the conditions of Corollary 3.1 may be realized by one IT, mn DFG's and one LT as shown in Figure 2.*

The nature of the definitions of the ideal elements is such that regularity is not required in Theorem 3.2. However, in practice, actual circuit building blocks corresponding to these

*One input line is assumed; if there are multiple input lines, one IT element for each line is required (see example).



a) digital multiplexor

b) digital function generator

c) ideal trigger element

d) logical transformation

FIG. 1. IDEAL CIRCUIT ELEMENTS

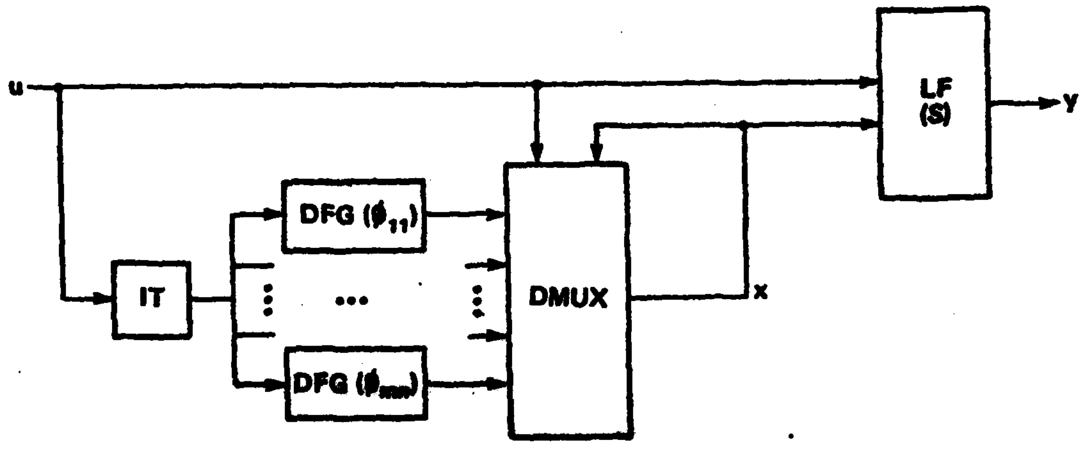


FIG. 2. REALIZATION OF SIMPLE ASYNCHRONOUS MACHINE (SAM)

ideal elements have built-in switching delays which limit their performance. The nature of these limitations becomes more apparent when the problem of synthesizing the DFG's in Figure 2 is considered. In addressing this problem, the relationship of the present theory with traditional state-transition based approaches will also become apparent; in fact, these approaches constitute a rather special case of the foregoing theory. This process requires a few more definitions (see Figure 3)!

Definition 3.11: An ideal resettable integrator (RI) has a binary-valued input b and a real-valued output, h defined by

$$h(t) = t - t_b$$

where t_b is the time of the most recent $\emptyset \to 1$ transition on b.

Definition 3.12: An ideal compurator (C) is a memoryless element with two real inputs h_1 and h_2 and a binary output, b, defined such that

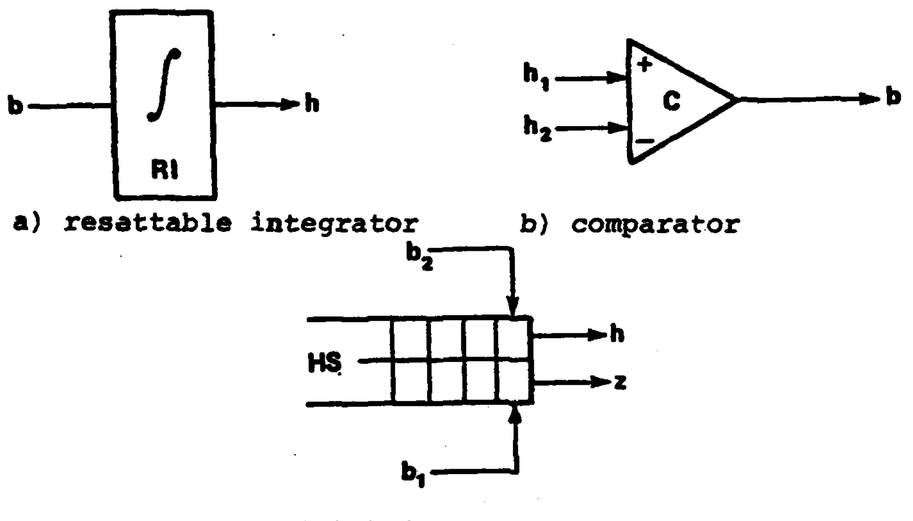
$$b(t) = \begin{cases} 1 & \text{whenever } h_1(t) \geq h_2(t) \\ 0 & \text{otherwise} \end{cases}$$

Definition 3.13: An ideal hybrid stack (HS) with data sequences $\{h_k\}$ and $\{z_k\}$, $k=\emptyset,1,\ldots$ has two binary inputs, b_1 and b_2, one real output h, and one discrete output $z \varepsilon Z$, a finite set. These are related as follows

$$h(t) = h_k \varepsilon R$$
$$z(t) = z_k \varepsilon Z$$

where k is the number of $\emptyset \to 1$ transitions of b_2 since the most recent $\emptyset \to 1$ transition of b_1, i.e., b_1 resets the stack pointer and b_2 increments the stack pointer.

Remark 3.8: The ideal hybrid stack can be approximated by a tandem interconnection of very long analog and digital shift registers.



a) resattable integrator b) comparator

c) hybrid stack element

FIG. 3. MORE IDEAL CIRCUIT ELEMENTS

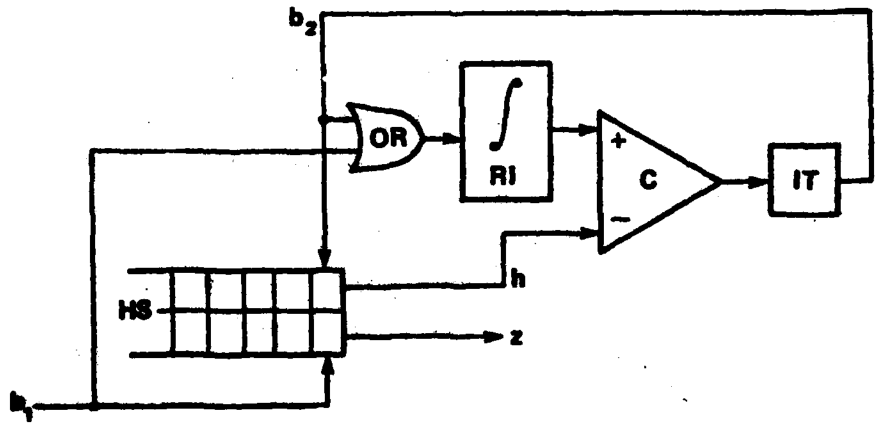


FIG. 4. REALIZATION OF DIGITAL FUNCTION GENERATOR IN FIGURE 2 (TIME-INVARIANT CASE)

Corollary 3.3: If the conditions of Theorem 3.2 hold and the SAM is regular, it may be represented as in Figure 2, with the DFG's synthesized as in Figure 4.

Since the SAM is regular, the number of transitions of any $\phi_{ij}(.,\emptyset)$ in Figure 2 is finite for any finite interval and countable on $[\emptyset,\infty)$. Let $\{h_k\}$ denote the sequence of intertransition times and let $\{z_k\}$ denote the sequence of output values (elements of X) of $\phi_{ij}(.,\emptyset)$. These are stored in the HS of Figure 4. The values z_k are delivered at the appropriate times by the clocking circuit consisting of the RI, C and IT, while b_1 (the input shown on Figure 2) correctly resets the circuit whenever a new input value occurs.

The obvious limitation of Corollary 3.3 is that the stacks (in general) have infinite memory requirements. However, there is great variety of interesting special cases in which the stack elements can be finitely generated. For instance, if the number of transitions for each ϕ_{ij} is finite, each corresponding hybrid stack can be terminated (formally) with an element $h_K=\infty$, $z_K=z_\infty$. Upon reaching this element, the comparator output of Figure 4 will never change state, and hence the output z_∞ will remain unless and until b_1 is retriggered. In this case, the hybrid stacks can be replaced with hybrid shift registers. A more interesting, and quite general, special case is where the elements of the stack may be recursively generated. This case corresponds to the one considered in [9] and [11]. Two final definitions are required for this case.

Definition 3.14: An ideal real recursive function (RRF) generator for the function $f:R\to R$ with kernel $h_\sigma \in R$ has two binary input lines b_1 and b_2 and one real-valued output line h. These are related as follows

$$h(t)=f^k(h_o)$$

where k is the number of $\emptyset \to 1$ transitions of b_2 since the most recent $\emptyset \to 1$ transition of b_1, and $f^k(.)$ denotes the k-th iterate of f. Thus, b_1 serves as a reset line and b_2 causes successive iterates to be generated.

Remark 3.9: The properties of iterates of certain functions have been studied by Klein and Kaliski [12]; they point out relations to ergodic theory and coding theory.

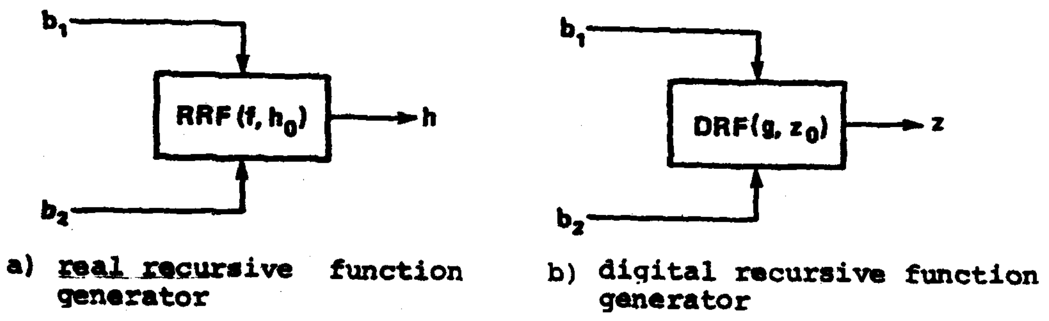
Definition 3.15: An ideal digital recursive function generator (DRF) for the function $g:X\to Z$ with kernel $z_o \varepsilon Z, Z=\{z^1 \ldots z^q\}$, has two binary input lines b_1 and b_2, and one discrete output $z \varepsilon Z$. These are related as follows

$$z(t)=g^k(z_o)$$

where k is the number of $\emptyset \to 1$ transitions on b_2 since the most recent $\emptyset \to 1$ transition of b_1, and $g^k(.)$ denotes the k-th iterate of g.

Remark 3.1Ø: In Definition 3.15, since Z is a finite set, $g^K=g$, for some finite K which is bounded by q^2. Thus, in the absence of resets, the sequence of values assumed by z in a DRF is necessarily periodic.

Corollary 3.4. There exist subsets of the regular SAMs which admit a finite parameterization. One such subset is termed the independently recursively generated SAM's (IRGSAM). These are characterized by the existence of independent functional recursions for the stack elements in Corollary 3.3 (Figure 4). The realization for these systems is shown in Figure 6.



a) real recursive function generator

b) digital recursive function generator

FIG. 5. COMPUTATIONAL ELEMENTS

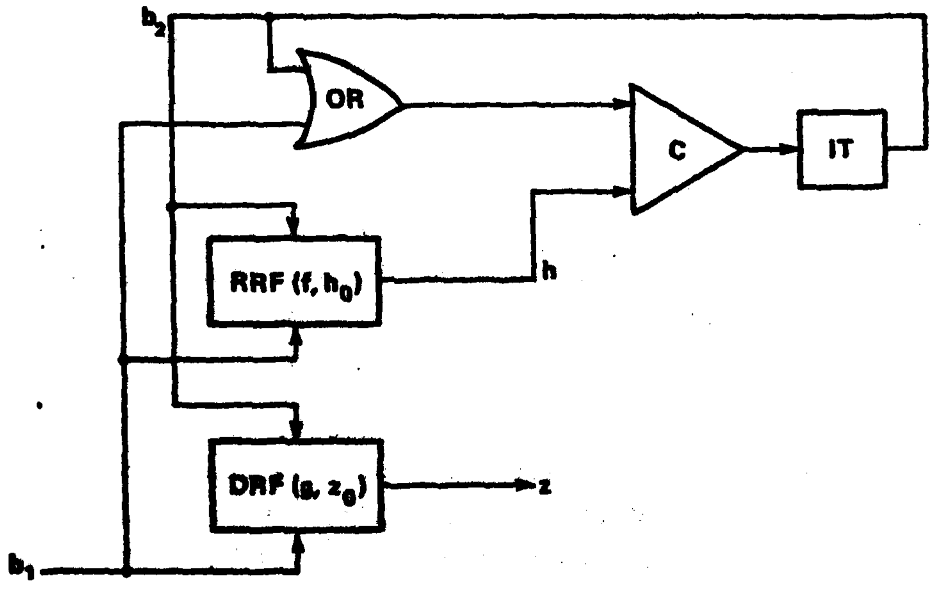


FIG. 6. REALIZATION OF INDEPENDENTLY RECURSIVELY GENERATED SIMPLE ASYNCHRONOUS MACHINE (IRGSAM)

IV. Example

As an example, one mode of operation of the Signetics 2651 Programmable Communications Interface^{tm} (PCI) is considered [13]. This is a universal synchronous/asynchronous data communications controller chip which accepts programmed instructions from a microprocessor and supports several serial communications disciplines, both synchronous and asynchronous. It serializes parallel data characters output from the microprocessor for transmission and can simultaneously receive serial data and convert it to parallel data for input to the microprocessor. It is packaged in a 28-pin DIP. A block diagram of the PCI is shown in Figure 7.

Since the PCI has 11 8-bit binary user-accessible registers, a rough bound on the number of possible states is 2^{88}. In the design of such devices, it is common to employ non-minimal(!) realizations to simplify the internal logic. As a special case, consider the PCI configured for as an asynchronous transmitter for 5-bit characters with no parity and two stop bits. In this mode, the input lines can be taken as

T_xC external transmitter clock (pin 9)
T_xEN transmit enable (command register, bit 0)
$\overline{CE}$ chip enable (pin 11)

and the relevant outputs are

T_xRDY transmitter ready (pin 15, and status register
bit 0)
T_xEMT transmitter empty (pin 18, and status register,
bit 2)
T_xD transmitted data (pin 19)

In this mode, $\overline{CE}$ causes parallel data to be clocked into the

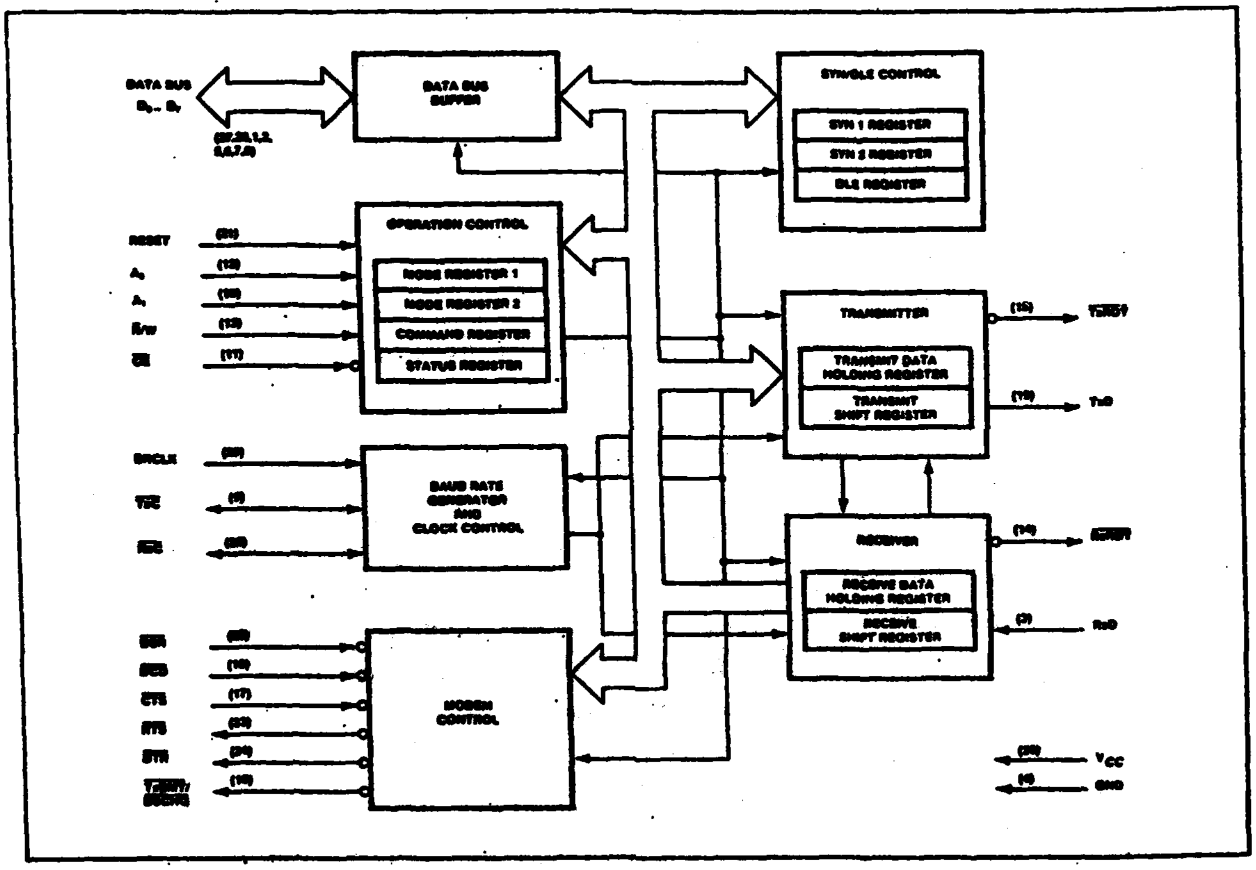


FIG. 7. BLOCK DIAGRAM OF SIGNETICS 2651 PROGRAMMABLE COMMUNICATIONS INTERFACE (PCI). MODIFIED FROM [13].

transmit data holding register (THR). When T_xEN is asserted by programming the command register (and $\overline{CTS}$ is low), transmission will commence and $\overline{TXRDY}$ will be asserted (low). Normally, the microprocessor will test this status bit, deposit new parallel data and pulse $\overline{CE}$, which resets $\overline{TXRDY}$ (high). However, if new data has not been received from the microprocessor by the time the last data bit is being set, $\overline{T_xEMT}$ is asserted (low) and marking bits are transmitted at the output. If data is subsequently received from the microprocessor, the end of the $\overline{CE}$ pulse resets $\overline{T_xEMT}$ and $\overline{T_xRDY}$; as soon as the data is clocked into the THR, then $\overline{T_xRDY}$ is reset (low) and the new data is transmitted, etc.

For convenience, the state set may be chosen from the timing diagram (Figure 8), to consist of states A, S_1,S_2,S_3,S_4, S_5,B,C, and D, where

A	=	start bit transmission
S_1	=	D_o transmission
S_2	=	D_1 transmission
S_3	=	D_2 transmission
S_4	=	D_3 transmission
S_5	=	D_4 transmission
B	=	stop bit 1 transmission
C	=	stop bit 2 transmission
D	=	mark transmission

Thus there are a total of $(2)^3 = 8$ input values and 9 state values, for a possible maximum of $8 \times 9 = 72$ state transition functions to be generated. However, most of these are trivial. Two examples will be considered: (1) the type of transition occurring between A and S_1, and (2) a transition between D and A after $\overline{T_xEMT}$ has been recently cleared by a data transmission.

<u>Case (1)</u>: This case is defined by an input transition to

TxRDY, TxEMT

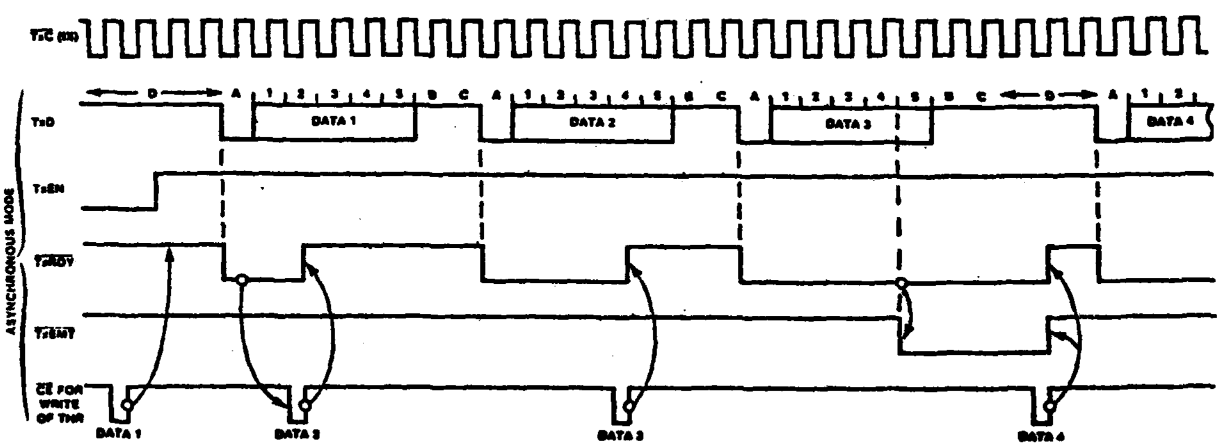


FIG. 8. TIMING DIAGRAM FOR ASYNCHRONOUS CHARACTER TRANSMISSION BY SIGNETICS 2651 PCI (5-BIT CHARACTERS, NO PARITY, 2 STOP BITS). MODIFIED FROM [13]

($\overline{T_x C}$, $T_x EN$, $\overline{CE}$) = (0,0,0) when the state is A. In this case, there is an almost immediate (650 nsec) transition to state S_1, and the outputs are ($\overline{T_x RDY}$, $\overline{T_x EMT}$, $T_x D$) = (0,1,D_0). This can be realized by the one-state timing circuit of Figure 6 which switches from A to S_1 at 950 nsec after the transition.

<u>Case (2)</u>: This case is defined by an input transition from (0,1,0) to (0,1,1) in state D, as shown at the end of the timing diagram. In this case, there is a significant delay while the holding register accepts the data and transfers it to the transmitter shift register for serialization. Thus the state transition function shifts from D to A after a time which may be more than one clock cycle. The (current) output is (1,1,1). This time delay may be realized as in Case (1), however one

should also proceed to consider the situation after the next clock cycle, as shown in the timing diagram, which illustrates the difference between a single input line having 8 values and 3 input lines having 2 values. For a single input line having 8 values, every new clock transition would reset this function generator, which is incorrect in the present case. With 3 input lines, only a transition on the $\overline{CE}$ line will reset this function generator, while transitions on the other input lines (in particular, $\overline{T}_x\overline{C}$) will leave it unaffected. (See footnote following (3.3).)

Most of the other state transitions are well-approximated as instantaneous, as in Case (1), and admit simplified realizations when this approximation is acceptable. The state transition functions are simple in this mode of operation because there is an external clock, i.e., generally, state transitions cannot occur without a clock transition, which is by definition an input transition. The state transition functions are defined for <u>constant</u> inputs, which is equivalent to predicting what happens if the clock <u>stops</u> following the input transition! The PCI admits another mode of operation, where the transmitter clock is taken from an internal baud rate generator. In this case, the state transition functions for constant inputs would switch spontaneously as dictated by the internal clock, and the realization would contain an internal clocking mechanism that was still practical to implement as described in Corollary 3.4.

V. Conclusions

The realization theory presented in Section 3 is based on semigroup theory and thus, unlike many common asynchronous machine representations, allows one to draw on a wealth of intuition about dynamic system theory. In fact, only the simplest case has been presented here: generalizations to hybrid-state and pulse modulated systems are clearly possible, for example. The results have some favorable properties which one would not necessarily expect: it is relatively easy to identify candidates for state variables for representative applications, and furthermore, the functions (ϕ_{ij}) required for the realization may often be readily determined from experiments on model systems. Finally, there is a significant class of systems which can be accurately realized by a finite number of elements and which correspond to practical digital devices. The theory provides a rigorous basis for the delineation of "control" (sequencing, and/or timing elements), and computational or "data flow" elements.

References

1. Aviation Week & Space Technology, Nov. 9, 1981, "Shuttle Launch Delay Attributed to Oil, Software Problems", pp. 20-21.

2. Young, K.D., and Kwatny, H.G., "Formulation and Dynamic Behavior of a Variable Structure Servomechanism", Proc. 1981 Joint Auto. Control. Conf., Paper WP-3F. (June, 1981, Charlottesville, VA.)

3. Wimpey, D.G., Johnson, T.L., and Kaliski, M.E., "Realization of A/D and D/A Coders", Proc. 1981 Joint Auto Contr. Conf., Paper WA-5A. (June 1981, Charlottesville, VA.)

4. Kohavi, Z., Switching and Finite Automata Theory, McGraw-Hill Publishing Co., Ltd., New Delhi, 1978.

5. Unger, S.H., Asynchronous Sequential Switching Circuits, J. Wiley & Sons, Inc., New York, 1969.

6. Miller, R.E., Switching Theory (Vol. II), J. Wiley & Sons, Inc., New York, 1966.

7. Peterson, J.L., "Petri Nets", ACM Computing Surveys, Vol. 9, No. 3, pp. 223-252, 1977.

8. Kalman, R.E., Falb, P.L. and Arbib, M.A., Topics in Mathematical System Theory, McGraw-Hill, New York, 1969.

9. Johnson, T.L., "Finite-State Control of Continuous Processes", Proc. 7th IFAC Congress, Helsinki, Finland, 1978.

10. Willems, J.C., and Mitter, S.K., "Controllability, Observability, Pole Allocatoin and State Reconstruction", IEEE Trans. Auto. Control, Vol AC-16, No. 6, pp. 582-596 (1971).

11. Johnson, T.L., "Analytic Models of Multitask Processes", Proc. 20th IEEE Conf. on Decision and Control, pp. 738-740, December 1981, San Diego, CA.

12. Klein, Q., and Kaliski, M., "Functinal Equivalence in a Class of Autonomous One-Dimensional Nonlinear Discrete Time Systems", Inform & Control, Vol. 42, No. 2, 1979.

13. Signetics, Inc., "Programmable Communications Interface (PCI)", Application Notes 2651-1, July 1978.

1983 AMERICAN CONTROL CONFERENCE

TOWARDS A THEORY OF ASYNCHRONOUS, REAL-TIME CODERS AND THEIR APPLICATIONS TO DISCRETE-CONTROL OF CONTINUOUS PROCESSES

TP5 - 3:45

by

Dr. Martin E. Kaliski* and Dr. David G. Wimpey

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115 USA

*Supported in Part by the United States Air Force Office of Scientific Research, AFSC, Contract F49620-82-C-0080

ABSTRACT

This paper develops a model of a real-time coder --a type of generalized A/D converter which maps sequences of n-tuples of real numbers into the binary alphabet {0,1}. Coders form an essential interface in the discrete-control environment between the continuous state plant to be controlled, and the discrete controller itself. These are partially-specified input/output maps, and finite-state realizations of them are considered. Examples of the developed theory are given.

I. Introduction

A key element in the theory of discrete-control of continuous systems is the coder--the (hybrid) interface between the continuous-valued part of the system and the discrete-valued part of the system (typically a microcomputer). (Wimpey, 1982; Kaliski and Lemone, 1980)

The coder is a kind of generalized analog-to-digital converter, which in the discrete-time case, maps sequences of elements of Rn, for some n>=1, into a finite set, which we may take as the binary alphabet {0,1}. An extensive study of coder design, using both algebraic methods (drawn from standard "finite" automata theory) and linguistic methods (based upon a concept of languages defined over real alphabets) has already been pursued by the authors in the synchronous case, as cited above. This case, wherein both the transitions of the discrete-time plant, and of the (finite-state) controller, occur in synchrony, under the regulation of an external clock, is clearly a special one, and many application contexts, where actions are triggered by events occuring asynchronously, do not fall under the umbrella of this model. Thus, in this paper, a theory of coders which are intrinsically asynchronous, and which explicitly incorporate the notion of time, is initiated.

We model asynchronous activity by using ideal "samples and holds" to record the exact moment in time at which events occur. This allows us to define what we will call "real-time coders." In their most elementary form they code strings of real numbers and are maps defined from a set S into {0,1} where S consists only of finite-length strings of pairs of the form (r,t), where "r" is a real number (produced by some process) and "t" is the time at which "r" is "seen." Because time always increases, the strings in S are always of the form

(r1,t1) ... (rk,tk)

where the t-values form a monotone increasing sequence and where the differences t2-t1, t3-t2,..., are always greater than some apriori lower bound TI.

One approach to the analysis and design of systems capable of implementing such maps is to view them as special forms of synchronous coders (in this case on R2) defined over specially restricted domains. The key element, then, in a successful theory is to develop ways of extending these real-time coders to all of R2 in a manner that allows well-behaved devices to be constructed. This paper will approach the problem of specifying the "don't-cares" of these maps so as to allow finite-state (finitary) realizations to be found. Several examples of real-time coders will be given.

Notation: For X a given set, X+ denotes the set of all finite-length non-null sequences of elements of X.

II. Generalizing the Simple Coder Model

Standard coders do not explicitly incorporate time into their structure. One way to introduce time into the coder model is by making it an explicit input to the coder--its value reflecting the "arrival" time of the other coder inputs. Doing this transforms a coder C:R+-->{0,1} into one which maps (RxR)+-->{0,1}. However, this new coder is defined only over a limited subset of (RxR) + -- namely those finite length strings of the form:

(r1,t1) ... (rk,tk)

with t1 < t2 < ... < tk, since time always moves forwards. (Due to practical restrictions inherent in physical implementations of coders, this strictly monotone sequence of time values is further constrained: there exists some real value TI such that, for j=2,..., k, tj - t j-1 >= TI. This reflects the fact that input changes cannot occur arbitrarily quickly.)

Let us refer to this class of coders as "real-time coders," which we will denote as RTC's.

III. An Example: A Real-Time Discrete Integrator

Consider the following "integrator-like" example. If the coder input is (r1,t1), then the output is 0. If the coder input is (r1,t1)...(rk,tk), with k>1, then the output is computed as follows: We form the sum:

r1(t2-t1) + r2(t3-t2) + ... r k-1 (tk-t k-1)

If the sum is greater than 0 the output is 1; otherwise it is 0.

One way of implementing such a coder is to extend its domain to all of (RxR)+ and to then employ previously developed techniques (Kaliski and Lemone, 1980; Wimpey, 1982). The most natural way of extending it is by just using the above rule for arbitrary strings (u1,v1) ... (uk,vk), regardless of the nature of the sequence v1,v2,...,vk. That is, if the sequence is of length 1, the output is 0; if its length is greater than 1, we compute the above sum and test its positiveness. If it is positive, then the output is 1; otherwise it is 0.

Now this extended I/O map can be realized by the customary techniques of Nerode equivalence. It is easy to see that the Nerode state-reduced model of this real time coder is the following: There is one state for each possible sum of the form:

ul(v2-vl) + ... + u k-1 (vk-v k-1) - uk vk

where the "uk" in these sums is the same. The starting state of the coder is the congruence class of the null string, which consists of strings whose sum, formed as above, is 0 and whose "uk" (last term) is 0. The next state and output maps are as follows: (Note that they are well-defined.)

Next state map:

[(ul,vl) ... (uk,vk)], (c,d) --->

[(ul,vl) ... (uk,vk) (c,d)]

Output map:

[(ul,vl) ... (uk,vk)], (c,d) --->

POS (ul(v2-vl) + ... + u k-1 (vk-v k-1)
+ uk (d-vk))

where POS(x) is 1 if and only if x is > 0.

A natural embedding of the state set into RxR exists--

[(ul,vl) ... (uk,vk)] --->

(ul(v2-vl) + ... + u k-1 (vk-v k-1)-uk vk,uk)

Using this embedding, a two-dimensional realization of this coder can be specified as follows:

Next state map:

(x,y), (c,d) ---> (x + (y-c)d, c)

Output map:

(x,y), (c,d) ---> POS(x + yd)

Starting state:

(0,0)

IV. The Key Theoretical Problem: Structured Extensions of Partially Specified I/O Maps

As the above example so clearly indicates, in order to effectively realize real-time coders, and to employ the already developed theory for synchronous coders, we must find a good way to extend the RTC from its constrained domain to all of (RxR)+. If this extension can be made in a way that "preserves" the intrinsic structure of the RTC, then we have effected the development of a useful design tool. We restrict our attention to finite-state extensions in this paper.

Towards this goal, let us begin our discussion with the following lemmas, drawn from the basic ideas of automata theory. The proofs of Lemmas 1 and 2 are straightforward and are omitted for lack of space.

Lemma 1: Let f:S --> {0,1} be a given I/O map defined on a subset S of {0,1}+. Suppose that there is a finite state machine M which realizes an extension g of f to {0,1}+. Then there is a finite partition P={P1,...,Pk} of S such that the following property is true, for all strings A and B in S:

If A and B are in the same block of P, and C in {0,1}+ is any string for which both AC and BC are in S, then AC and BC are also in the same block of P (which is not necessarily the block that A and B are in), and f(AC)=f(BC).

A much more powerful "converse" to Lemma 1 is also true when S has the property that "once a string leaves "S" it never "gets back in," i.e., if A in {0,1}+ is not in S, then AB is not in S for any B in {0,1}+. Call such S (for lack of a better expression) "tightly structured."

Lemma 2: Let S be a tightly structured subset of {0,1}+ and P a partition {P1,...,Pk} of S, for which the partition property holds for f:S --> {0,1}. Then there exists an extension g of f to all of {0,1}+ which is finite state realizable.

Thus, we have:

Theorem 1: (Fundamental Realizability Theorem for Partially Specified I/O Maps)

Let S be a tightly structured subset of {0,1}+. Let f be an I/O map from S into {0,1}. Then there exists an extension g of f to all of {0,1}+ which is finite-state realizable if and only if S obeys the "partition property" with respect to f:

> "There exists a finite partition P={P1,...,Pk} of S such that for any j=1,...,k, and any A and B in Pj, if C in {0,1}+ is such that AC and BC are both in S, then AC and BC belong to the same block Pm of P as well (m not necessarily equal to j). Furthermore, f(AC)=f(BC)."

V. Generalizing to I/O Maps on RxR

All of this generalizes to maps on RxR, using the concept of a finite-automaton defined over a real alphabet (Wimpey, 1982). In particular it generalizes to Real-Time Coders, where, by the very nature of the domains of such coders (the set S), the tight structure criterion is met. (The input to a RTC goes "bad" as soon as the time-coordinate does not advance sufficiently far; once it becomes bad it remains bad).

We begin with an intuitive definition of a deterministic finite-state real automaton (FSRA). M is a FSRA if there exist a finite number of states Q1,..., Qk, and, associated with each state Qj, j=1,...,k, a partition Pj of the real numbers. A single state of M is labelled as the starting state of M, and certain states of M are classified as accepting states. Thus M is naturally associated with a coder CM: R --> {0,1}. Clearly all of this generalizes to inputs that are in RxR. We call such a finite-state automaton a 2-dimensional FSRA, and, for brevity denote such a device by the symbols 2DFSRA.

The notion of a non-deterministic FSRA also generalizes in a straightforward manner. In particular if one associates with each state Qj a cover Cj of R (or RxR) then one has a non-deterministic, completely specified FSRA. The proof that such a device is always equivalent to a deterministic FSRA is virtually identical to the one used in finite automata theory. We omit it.

Lemmas 1 and 2 immediately carry over for real time coders, as their domains are tightly structured. Specifically, let us formally define a RTC: (This definition complements the informal one given in section II above.)

Definition: A Real-Time Coder (RTC) is any map from S into {0,1} where S is the following subset of RxR:

S = { (r1,t1) ... (rk,tk) such that

t2-t1>TI,..., tk- t k-1 > TI }

where TI is some fixed real constant.

Note that S is tightly structured. There is nothing in the proofs of Lemma 1 and 2 that depends upon the fact that the inputs to the automata are 0 or 1. Thus the Lemmas generalize to the real-time coder case and, thus, we have:

Theorem 2: (Fundamental Finite-State Realizability Theorem for Real-Time Coders)

"Let RTC be a given real-time coder. Then RTC is finite-state realizable if and only if there exists a finite partition {P1,...,Pk} of its domain S such that for any j=1,...,k, and any A, B in Pj, if C in RxR is such that AC and BC are both in S then AC and BC are both in the same block Pm of the partition (m not necessarily equal to j) and RTC(AC)=RTC(BC)."

There is an explicit mechanism for constructing the finite-state realization when the partition property holds (as in the proof of Lemma 2.)

V. An Example of a Finitary Real-Time Coder

We give here a simple example based upon Wimpey (1982) to illustrate the concepts above. A finitary coder is one realizable by a 2DFSRA. Note that the real-time integrator considered earlier is definitely not finitary.

Consider the following real-time coder.

RTC((r1,t1)) = 0 if (r1+t1) >= 0; if not

RTC((r1,t1)...(rk,tk)) =

0 if (rk+tk) >= 0 and the number of non-negative sums r1+t1,..., r k-1 + t k-1 is even or zero.
1 otherwise

Is it finite-state realizable? And, if so, what is a realization for it? We proceed to answer these questions.

The domain S of this coder satisfies the partition property for the following partition {P1, P2, P3}:

P1 = {strings having an odd number of non-negative sums with the last sum negative}

P2 = {strings having an odd number of non-negative sums with the last sum non-negative}

and

P3 = {strings having an even number (or zero) non-negative sums}

By our developments above, then, RTC is finitary. In fact it is easy to directly construct a 2DFSRA for RTC, based upon the Lemmas above. Rather than do so directly it is clear that, as with finitary, ordinary coders, RTC may be realized *as a cascade of a simple* quantizer and a finite state machine: (Wimpey, 1982)

The quantizer Q outputs 0 if rj+tj is >=0; otherwise it outputs 1. The finite state machine M has three states corresponding to P1, P2, and P3. Call them Q1, Q2, and Q3. Q3 is the starting state. The accepting states are Q1, and Q3. The transitions are as follows:

(Q1,0) --> Q3 (Q1,1) -->Q1

(Q2,0) --> Q3 (Q2,1) -->Q1

(Q3,0) --> Q2 (Q3,1) -->Q3

VI. Conclusions

The process of "creatively" selecting "don't care" values so as to provide for meaningful extensions of real-time coders is a fundamental open research problem of this theory. Many more general coder types exist, such as the shift-finitary coders (Wimpey, 1980). Current work, then, is seeking to extend the above preliminary results to these more general coders, as well as to look at other ways of modelling asynchrony in coder design.

VII. References

Kaliski, M. E. and Lemone, K., "Discrete-Codings of Continuous Valued Signals," 1980 Conference on Information Sciences and Systems, Princeton University, Princeton, NJ.

Wimpey, D. G., "Finite-State Control of Discrete-Time Continuous Processes: An Automata-Motivated Approach," Ph.D. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1982.

Wimpey, D. G., Johnson, T. L., and Kaliski, M. E., "Realization of A/D and D/A Coders," 1981 JACC, Charlottesville, VA, June 1981.

MEMORANDUM: JANUARY, 1983

TOWARDS A THEORY OF FINITARY ASYNCHRONOUS CODERS

1) Generalizing the Simple Coder Model

The standard coders previously considered by us did not explicitly incorporate time into their structure. Rather, time was implicit, synchronous, and "regular". It made sense to talk about the kth input or output (or state), and the value of k automatically stepped through the integers.

About the simplest way to introduce time into the coder model is by making it an explicit input to the coder -- its value reflecting the "arrival" time of the other coder inputs. Doing this transforms a coder $C:R+\dashrightarrow\{0,1\}$ into one which maps $(R\times R)+\dashrightarrow\{0,1\}$. However, this new coder (call it RTC for "real-time coder") is defined only over a limited subset of $(R\times R)+$ -- namely those finite length strings of the form:

$(r1,t1) \ldots (rk,tk)$

with $t1 < t2 < \ldots < tk$, since time always moves forwards.

Due to practical restrictions inherent in physical implementations of coders, this strictly monotone sequence of time values is further constrained: there exists some real value TI (for "Temporal Input" constraint) such that, for $j=2,\ldots,k$, $tj - t\ j\text{-}1 >= TI$.

(Theoretically, at least, such constraints are ergodic-theoretic in nature -- a notion to be explored in greater depth during the remainder of this contract year.)

The output of the RTC may still be regarded as a binary number $b = 0$, or 1, and may be viewed as occuring within TO seconds after the last input arrives. (TO for "Temporal Output" constraint). Note that various racing problems, and hazards, and such, must be dealt with to physically implement such a device. For example, TO may have to be smaller than TI to assure that the correct output is read.

2. An Example: A Real-Time Discrete Integrator

Consider the following "integrator-like" example. If the coder input is (r1,t1), then the output is 0. If the coder input is $(r1,t1) \ldots (rk,tk)$, with $k>1$, then the output is computed as follows: We form the sum:

$r1(t2-t1) + r2(t3-t2) + \ldots\ r\ k\text{-}1\ (tk-t\ k\text{-}1)$

If the sum is greater than 0 the output is 1; otherwise it is 0.

One way of implementing such a coder is to extend its domain to all of (RxR)+ and to then employ the standard techniques we have already developed. The most natural way of extending it is by just using the above rule for arbitrary strings (u1,v1) ... (uk,vk), regardless of the nature of the sequence v1,v2,...,vk. That is, if the sequence is of length 1, the output is 0; if its length is greater than 1, we compute the above sum and test its positivity. If it is positive, then the output is 1; otherwise it is 0.

Now this extended I/O map can be realized by the customary techniques of Nerode equivalence. Let us see what the Nerode equivalence relation is in this case. First, consider two strings of length greater than 1. Two strings (u1,v1)...(uk,vk) and (p1,q1)...(pm,qm) will be equivalent if and only if the two sums below are equal for any (a1,b1)...(aj,bj):

u1(v2-v1) + ... + u k-1 (vk-v k-1) + uk(b1-vk)

+ a1(b2-b1) + ... + a j-1(bj-b j-1) and

p1(q2-q1) + ... + p m-1 (qm-q m-1) + pm(b1-qm)

+ a1(b2-b1) + ... + a j-1(bj-b j-1)

(Note that, strictly speaking, we want POS of each sum to be equal, where POS is the map POS(x)=1 if and only if x is positive; otherwise it is 0. It should be clear, however, that if the sums differ, we can append yet another term (a j+1 b j+1) which can cause differences in signs of these new sums, and thus differences in the POS-values.)

If j=1 we may ignore the second line in the two expressions above, and, in fact, may ignore its role in the equivalence condition completely, as it is identical in both expressions. Thus the two original strings will be equivalent if and only if the two first lines above are equal, for any b1 whatsoever. Setting b1 to 0, in particular, forces equality of the following two expressions:

u1(v2-v1) + ... + u k-1 (vk-v k-1) - uk vk and

p1(q2-q1) + ... + p m-1 (qm-q m-1) - pm qm

Let us denote this common value by A. Then we must also have that A+uk b1 = A+pm b1, for any real number b1. Since b1 may be 1, it immediately follows that uk must in fact equal pm.

Summarizing, then, a necessary and sufficient condition for two strings (u1,v1)...(uk,vk) and (p1,q1)...(pm,qm) of length greater than one to be Nerode equivalent is that uk=pm and that

u1(v2-v1) + ... u k-1 (vk-v k-1) - uk vk =

p1(q2-q1) + ... p m-1 (qm-q m-1) - pm qm

Now let us turn to strings of length 1. First suppose that we have two strings, both of length 1. Call them (u1,v1) and (p1,q1). It must be that for any (a1,b1)...(aj,bj), j>=1,

u1(b1-v1) + a1(b2-b1) + ... a j-1(bj-b j-1) =

p1(b1-q1) + a1(b2-b1) + ... a j-1(bj-b j-1)

where the terms on the right are not present if j=1. It is immediate that we must have -u1 v1 = -p1 q1, and u1=p1.

For one string of length 1, and the other of length greater than one, it can similarly be deduced that the requisite condition for equivalence is that, calling the strings (u1,v1) and (p1,q1)...(pm,qm):

u1=pm and

-u1 v1 = p1(q2-q1) + ... + p m-1 (qm-q m-1) - pm qm

As for the null string NL, a similar condition is derived: NL is equivalent to (u1,v1)...(uk,vk), k>=1 if and only if:

uk=0 and

u1(v2-v1) + ... u k-1(vk-v k-1) = 0

All of this suggests that the Nerode state-reduced model of this real time coder is the following. There is one state for each possible sum of the form:

u1(v2-v1) + ... + u k-1 (vk-v k-1) - uk vk

where the "uk" in these sums is the same. The starting state of the coder is the congruence class of NL, the null string, which consists of strings whose sum, formed as above, is 0 and whose "uk" (last term) is 0. The next state and output maps are as follows: (Note that they are well-defined)

Next state map:

[(u1,v1) ... (uk,vk)], (c,d) --->

[(u1,v1) ... (uk,vk) (c,d)]

Output map:

[(u1,v1) ... (uk,vk)], (c,d) --->

POS(u1(v2-v1) + ... + u k-1 (vk-v k-1)

+ uk (d- vk))

A natural embedding of the state set into RxR exists --

[(u1,v1) ... (uk,vk)] --->

(u1(v2-v1) + ... + u k-1 (vk-v k-1)-uk vk,

uk)

Using this embedding, a two-dimensional realization of this coder can be specified as follows:

Next state map:

(x,y), (c,d) ---> (x + (y-c)d, c)

Output map:

(x,y), (c,d) ---> POS(x + yd)

Starting state:

(0,0)

A diagram of this coder is shown in Figure 1. Note that it is assumed in this diagram that the inputs are connected to event-driven ideal samples and holds. Events occur at the input times, and the samples and holds serve to "capture" the inputs at these times, and the times themselves. In this diagram we assume that the output is read TO seconds after the input occurs.

3. The Key Theoretical Problem: Structured Extensions of Partially Specified I/O Maps

As the above example so clearly indicates, in order to effectively realize real-time coders, and to employ the already developed theory for synchronous coders, we must find a good way to extend the RTC from its constrained domain to all of (RxR)+. If this extension can be made in a way that "preserves" the intrinsic structure of the RTC, then we have effected the development of a useful design tool.

Towards this goal, let us begin our discussion with the following lemmas , drawn from the basic ideas of automata theory:

Lemma 1: Let f:S --> {0,1} be a given I/O map defined on a subset S of {0,1}+. Suppose that there is a finite state machine M which realizes an extension g of f to {0,1}+. Then there is a finite partition P={P1,...,Pk} of S such that the following property is true, for all strings A and B in S:

If A and B are in the same block of P, and C in {0,1}+ is any string for which both AC and BC are in S, then AC and BC are also in the same block of P (which is not necessarily the block that A and B are in), and f(AC)=f(BC).

Proof of Lemma 1: We may assume without loss of generality that M is a Moore machine; call its starting state Q1. Call the remaining states of M Q2,...,Qk. Define the partition P as follows: Block Pj of P, j=1,...,k consists of those strings of S which take state Q1 to state Qj. Clearly {P1,...,Pk} partitions S. If A and B are strings in S that are in the same block of P, say Pj, then for any C whatsoever, AC and BC are in the same block of P. In are in S. Furthermore, by the very definition of "M realizes g" it must be that g(AC)=g(BC) as AC and BC lead M to the same state. When AC and BC are both in S the g in the above equality may be replaced by f since g is an extension of f. QED

A much more powerful "converse" to Lemma 1 is also true when S has the property that "once a string leaves S" it never "gets back in", i.e. if A in {0,1}+ is not in S, then AB is not in S for any B in {0,1}+. Call such S (for lack of a better expression) "tightly structured".

Lemma 2: Let S be a tightly structured subset of {0,1}+ and P a partition {P1,...,Pk} of S, , for which the partition property holds for f:S --> {0,1}. Then there exists an extension g of f to all of {0,1}+ which is finite state realizable.

Proof of Lemma 2: Consider the non-deterministic finite-state Moore machine M defined as follows. The starting state of M will be called Q0. There is a special state of M (where strings not in S will take M) that will be called QNS. The remaining states of M correspond to the number of blocks in P in the following way. For j=1,...,k, look at Pj. If all the strings in Pj have that property that f maps them to 0, then create just a single state Qj0; if all the strings have the property that f maps them to 1, then create just a single state Qj1. Otherwise create two states Qj0 and Qj1. Thus M contains at least k+2 states, and at most 2k+2; in any case it is finite state.

The accepting states of M will be the Qj1's, if present, for j=1,...,k. The transitions of M are defined as follows: First let's look at Q0. If 0 is in S, then 0 is in some block of P, say Pj. Direct a 0-arrow from Q0 to Qj0 if f(0)=0; otherwise direct the 0-arrow to Qj1. If 0 is not in S then direct the 0-arrow to QNS. The 1-arrow from Q0 can similarly be constructed, based upon whether or not 1 is in S, and, if so, what the value of f(1) is. Next we consider QNS. This will be considered to be a "dead state" in the sense that once entered, it cannot be left. Thus both the 0-arrow, and the 1-arrow, go from QNS to QNS. (As we will see below, all strings that are not in S lead to QNS; by the tight structure constraint on QNS, once QNS is entered it is appropriate to get trapped there.)

Finally we turn to the transitions for the remaining states. Consider, for a given j, the states Qj0 and Qj1, or, if just one is present, whichever it happens to be. Consider the problem of drawing the 0-arrow. Now the strings in Pj divide into two classes -- those which, when followed by 0, are still in S, and those which, when followed by 0 are no longer in S. By the partition property on S, all of the strings so "continuable" by 0 are elements of the same block of P, say Pm, and the continued strings have the same f-value. We can use the above remarks to draw 0-arrows from Qj0 and/or Qj1 as follows: Suppose that there is a state Qj0. This means that there are strings in Pj for which f is 0. If any of these strings are "continuable" by 0, by the above remark, they all have the f-values when continued, and all belong to Pm, when continued. Call the f-value of the continued strings "b" (b=0,1). Clearly, by our very construction, state Qmb must be present. Draw a 0-arrow from Qj0 to Qmb. If Pj also contains strings for which f is 0, but which are not continuable, draw a 0-arrow from Qj0 to QNS. Thus one or two 0-arrows will emanate from Qj0, going to either QNS, Qmb, or both. A similar procedure can be used to draw 0-arrow(s) from Qj1. Furthermore the same procedure can be used to draw the 1-arrows from Qj0 and Qj1.

It is easy to see that if a1,...,an is a string in S that (by the tight structure of S), each of the strings a1, a1 a2, ... , a1,....,a n-1 is in S an that there is a path in M, starting at Q0, and remaining thereafter among the Qj's. It is also easy to see that M accepts a1,...,an in S if and only if f(a1,...,an) is [illegible]. [illegible] accept strings that are not in S, in general, but the only strings in S that are accepted are those for which f is 1.

Now M can be converted to an equivalent deterministic Moore machine M' which accepts exactly the same strings as does M. Let us define g:{0,1}+ --> {0,1} by g(A)=1 if and only if M' accepts A. By our remarks in the preceding paragraph, g is an extension of f. Obviously g is finite-state realizable (by M'). The proof is complete. QED

We may summarize the above two Lemmas with the following Theorem, which we call the Fundamental Realizability Theorem for Partially-Specified I/O Maps:

Theorem 1: (Fundamental Realizability Theorem for Partially Specified I/O Maps)

Let S be a tightly structured subset of {0,1}+. Let f be an I/O map from S into {0,1}. Then there exists an extension g of f to all of {0,1}+ which is finite-state realizable if and only if S obeys the "partition property" with respect to f:

" There exists a finite partition P={P1,...,Pk}

of S such that for any j=1,...,k, and any A and B

in Pj, if C in {0,1}+ is such that AC and BC are

both in S, then AC and BC belong to the same block

of Pm of P as well (m not necessarily equal to k).

Furthermore f(AC)=f(BC)."

Generalizing to I/O maps on RxR: ________________

In the pages that follow we will show that all of this generalizes to maps on RxR, using a form of our earlier definition of finite-automaton defined over real alphabets. In particular this will generalize to Real-Time Coders, where, by the very nature of the domains of such coders (the set S), the tight structure criterion is met. (The input to a RTC goes "bad" as soon as the time-coordinate does not advance sufficiently far; once it becomes bad it remains bad).

We will culminate, therefore, in the following variant of Theorem 1:

Theorem 2: (Fundamental Realizability Theorem for Real-Time Coders)

Let RTC be a real-time coder. Then RTC is finite-state realizable (in the finitary coder sense) if and only if its domain obeys the partition property.

We begin we an intuitive definition of a deterministic finite-state real automaton (FSRA). M is a FSRA if there exist a finite number of states Q1,...,Qk, and, associated with each state Qj, j=1,...,k, a partition Pj of the real numbers. A single state of M is labelled as the starting state of M, and certain states of M are classified as accepting states. Thus M is naturally associated with a coder CM: R --> {0,1}. Clearly all of this generalizes to inputs that are in RxR. We call such a finite-state automaton a 2-dimensional FSRA, and, for brevity denote such a device by the symbols 2DFSRA.

The notion of a non-deterministic FSRA also generalizes in a straightforward manner. In particular if one associates with each state Qj a cover Cj of R (or RxR) then one has a non-deterministic, completely specified FSRA. The proof that such a device is always equivalent to a deterministic FSRA is virtually identical to the one used in finite automata theory. We sketch it below.

One creates a deterministic FSRA by initially defining a state for each of the non-empty subsets of {Q1,...Qk}. The starting state of this new machine is the same state (singleton subset) as that of the non-deterministic one. The accepting states are any states whose corresponding subsets contains at least one accepting state of the non-deterministic machine. To define the transitions of a given state we look at each state in the associated subset. We pick a real number, say r, and ask to what states in the original non-deterministic machine r takes us. We take the union of all of these r-mapped states (for each state in the associated subset) and that subset of {Q1,...,Qk} is where we send r to. We repeat this for each real number, thereby associating a partition of the reals (or of RxR in the two-dimensional case) with each subset of {Q1,...,Qk}. The rest of the proof is clear. We can easily demonstrate that the deterministic machine and the non-deterministic one accept exactly the same set of strings.

The proofs of Lemmas 1 and 2 immediately carry over for real time coders, as their domains are tightly structured. Specifically, let us formally define a RTC:

Definition: A Real-Time Coder (RTC) is any map from S into {0,1} where S is the following subset of RxR:

S = { (r1,t1) .... (rk,tk) such that

t2-t1>TI, ..., tk- t k-1 > TI }

where TI is some fixed real constant.

Note that S is tightly structured. There is nothing in the proofs of Lemmas 1 and 2 that depends upon the fact that the inputs to the automata are 0 or 1. Thus the Lemmas generalize to the real-time coder case and, thus, Theorem 2 holds. We repeat its statement below and give, in section 2.1, a detailed example.

"Let RTC be a given real-time coder. Then RTC is

finite-state realizable if and only if there exists

a finite partition {P1,...,Pk} of its domain S such

that for any j=1,...,k, and any A,B in Pj, if C

in RxR is such that AC and BC are both in S then AC

and BC are both in the same block Pm of the partition

(m not necessarily equal to j) and RTC(AC)=RTC(BC)."

There is an explicit mechanism for constructing the finite-state realization when the partition property holds.

4. An Example of a Finitary Real-Time Coder

We give here a simple example based upon Wimpey (1982) to illustrate the concepts above.

Consider the following real-time coder.

RTC((r1,t1)) = 0 if (r1+t1)>=0; 1 if not

RTC((r1,t1)....(rk,tk)) =

0 if (rk+tk)>= 0 and the number of

non-negative sums r1+t1,...,

r k-1 + t k-1 is even or zero.

1 otherwise

Is it finite-state realizable? And, if so, what is a realization for it? We proceed to answer these questions.

The domain S of this coder satisfies the partition property for the following partition {P1,P2,P3}:

P1 = { strings having an odd number of non-negative sums with the last sum negative}

P2 = { strings having an odd number of non-negative sums with the last sum non-negative} and

P3 = { strings having an even number (or zero) non-negative sums}

Note that if strings A and B in P1 are such that AC and BC are in P1 then RTC(AC)=RTC(BC)=1; if they are in P2 then RTC(AC)=RTC(BC)=0. If AC and BC are in P3 then RTC(AC)=RTC(BC)=1. Similarly if A and B are in P2 then if AC and BC are in P1 the common output value is 1; if they are in P2 the common output value is 0. If they are in P3 the output is 1. Finally if A and B are in P3 then the common outputs are (for termination in P1,P2, and P3 respectively) 1,0,and 1.

By our developments above, then, RTC is finitary. In fact it is easy to directly construct a 2DFSRA for RTC, based upon the arguments above. Rather than do so directly it is clear that, as with finitary, ordinary coders, RTC may be realized as a cascade of a simple quantizer and a finite state machine:

The quantizer Q outputs 0 if rj+tj is >=0; otherwise it outputs 1. The finite state machine M has three states corresponding to P1,P2, and P3. Call them Q1,Q2, and Q3. Q3 is the starting state. The accepting states are Q1, and Q3. The transitions are as follows:

(Q1,0) --> Q3 (Q1,1) -->Q1

(Q2,0) --> Q3 (Q2,1) -->Q1

(Q3,0) --> Q2 (Q3,1) -->Q3

A THEORY OF ORBITAL BEHAVIOR IN A CLASS OF NONLINEAR SYSTEMS: "CHAOS" AND A SIGNATURE-BASED APPROACH

by M.Kaliski, Northeastern University
Boston, MA *
S.Yunkap Kwankam, Universite de Yaounde,
Yaounde, CAMEROUN
P. Halpern, Northeastern University,
Boston, MA

(* This author's work has been supported in part by the United States Air Force Office of Scientific Research, AFSC, Contract Number F49620-82-C-0080.)

Mailing Address: Dr. Martin E. Kaliski
Professor and Director of Computer Engineering
405 Dana
Northeastern University
Boston , MA
02115

submitted to

International Journal

of Systems Science

Abstract

A theory of orbital behavior in certain autonomous one-dimensional nonlinear systems is pursued, using a approach based upon the concept of orbital signature. Particular attention is paid to the fixed point structure of such systems with the ultimate aim of using the signature repertoires of these systems to characterize fixed-point orders and the presence of "chaotic regimes". A system-theoretic approach is pursued here -- an approach which complements other recent studies of a more analytical nature (employing ergodic theoretic methods.) Chaotic behavior in a certain subclass of these systems is completely characterized in terms of the first two iterates of a specific known point in the range of the system transition function.

Table of Notation

symbol	meaning
fpq, f	a generic unimodal or subbell function with breakpoint p and peak value q
fk pq, fk	for k>=0, the kth iterate of fpq or f
sigf(x),sig(x)	the signature (infinite) of x under the mapping f
sigfk(x),sigk(x)	the k-signature of x under the map f
ls(x), rs(x)	the left, right signatures of x
lsk(x), rsk(x)	the left, right k-signatures of x
\	such that
Sk	the k-signature repertoire of a

	given map
S	the (infinite) signature repertoire of a given map
.	sequence concatenation
^	set intersection
a**n	the sequence a a a ... a (n times)
U	set union
s*j	the jth left rotation of s
Ak, A	the signature bins of a given string s
Lj(s)	the jth left shift of s
[...]	closed interval
[	subset of
<>	not equal to

I. Introduction

There has been great interest in recent years in the autonomous behavior of nonlinear one-dimensional systems defined over the unit interval [0,1] {Kaliski and Klein, 1982; Klein and Kaliski, 1979; Collet and Eckmann, 1980; Li and Yorke, 1975; Parry, 1966; Baillieul , Brockett, and Washburn, 1980; and Feigenbaum, 1980, to name several} . There is, in particular, interest in the "chaotic behavior" of the equilibrium (fixed) points of such systems. This "random" behavior arises even in processes describable by simple first-order difference equations of the form:

$$x\ k+1 = f(xk)$$

where $f:[0,1]\text{-->}[0,1]$.

Various approaches have been used to describe systems of the above form, ranging from graphical methods {May and Oster,1976} , to purely analytical techniques {Li and Yorke,1975} and ergodic theoretic appoaches {Parry,1966}.

A system theoretic approach, introduced by Klein and Kaliski, and cited above, is pursued in this paper. This approach treats the function f as the state transition function of an autonomous one-dimensional nonlinear discrete-time system. The concept of "signature" (defined below) is used to describe the orbit of any given starting state.

This paper serves to characterize the fixed point structure of such systems through the use of signature repertoires. Our development is somewhat detailed because of the need to formalize and define our "working tools"; nonetheless, our results are of intrinsic interest because:

(i) they characterize "chaotic behavior" (the presence of fixed points of all periods) in a subclass of such systems strictly in terms of the first two iterates of a specific point in the range of the system transition function.

(ii) they demonstrate the utility of the signature concept, a concept which complements alternate approaches based upon ergodic theory and other measure theoretic concepts.

Much of this material appeared in somewhat different format in one of the authors' doctoral dissertations. [Kwankam, 1979]

II. Basic Concepts: Subbells, Signatures, and Gray Code Order

As much of this development is based upon the concepts introduced by the earlier work of Kaliski and Klein, we merely summarize their basic ideas in the paragraphs below.

II,1 We begin by describing the types of functions considered in this paper.

A subbell function is a continuous map $f:[0,1] \to [0,1]$ for which (Figure 1)

(i) $f(0) = f(1) = 0$,

(ii) f has a unique maximum q at some point p in $(0,1)$

and

(iii) f is strictly monotone on $[0,p]$ and on $[p,1]$.

We shall refer to p as the breakpoint of f and denote by fpq a subbell map f with breakpoint p and peak value q. Similarly, for k a positive integer, $fk\ pq$ denotes the kth iterate of fpq (i.e. the k-fold composition of fpq with itself).

A subbell function is a special type of "unimodal" function:

A unimodal function is a continuous map $f: [0,1] \to [0,1]$ for which conditions (ii) and (iii) above hold, but not necessarily condition (i). (We extend the notation fpq and fk pq to unimodals in general.) (Figure 2)

II,2 We introduce the concept of (orbital) signature next. Let x in [0,1] be given, k a positive integer. The k-signature of a sequence under the mapping fpq, denoted by sigfk(x) , is the length k string:

b0 b1 b2 ... b k-1

where for i= 0,...,k-1,

	bi =	0	if 0 <= fi pq(x) < p
	=	-	if fi pq(x) = p
and	=	1	if p < fi pq(x) <= 1

(where f0 pq(x) = x, by convention)

s = b0 b1 b2 ... b k-1 will be called regular if, for all i >= 0, bi = 0 or 1 but not - . If s is not regular it will be called irregular . In a similar fashion one can define the (infinite) signature sigf(x) of x by simply letting "k" range over all the positive integers. The notions of regularity and irregularity generalize in a straightforward way. In the sequel when the word signature is used without further qualification it may mean either finite length or infinite signature, according to the context of the discussion. Further, when the specific function fpq is clear from this context we will often simply write "f" and drop the f from "sigf". When a point x has a regular signature x is said to be regular; otherwise it is said to be irregular. Note that if x is regular then for all k, every k-signature of x is regular, and conversely.

II,3 We can define a total ordering relationship upon binary strings (not necessarily regular signatures) which the reader will recognize as Gray Code order:

Let s1 = b0 b1 ... and s2 = d0 d1 ... denote two given binary sequences of equal finite length, or both of infinite length. Then s1 < s2 if

(i) s1 is not equal to s2, and

(ii) if bit position j is the first one at which s1 and s2 differ, then

b0 + ... + b j-1 = 0 and

$d0 + \ldots + d_{j-1} = 1$

where + is the Exclusive OR function.

This ordering is fundamental in the theory of signatures, in that all unimodal functions obey a "monotonicity of signatures property" (monotone with respect to this ordering.) This is explored below in section II,5.

II,4 Let s be an irregular infinite signature of some point x under a given unimodal map. An instance of s is any binary string obtained from s by arbitrarily inserting 0's or 1's for each - in s. We refer to the least (in the < ordering) instance of s as the left-signature of s, ls(x); we similarly define the right-signature of s, rs(x), as the greatest instance of s. The notion of left and right signature trivially extends to regular points x, where ls(x) = rs(x) = sig(x). Note that these definitions hold equally well for finite signatures, where we write the left and right k-signatures of x as lsk(x), and rsk(x), respectively.

We define the concept of the exploded k-signature repertoire, denoted by Sk, to be

Sk = {rsk(x) \ x in [0,1]} U {lsk(x) \ x in [0,1]}

Thus the exploded k-signature repertoire of f consists of all k-signatures of regular points, and the greatest and least k- signatures of all irregular points (hence the irregular k-signatures are "exploded".) A similar interpretation holds for the (k=infinity) exploded signature repertoire of f, denoted by S.

II,5 As indicated above all unimodal functions obey a fundamental monotonicity property with respect to their signatures. We cite the property without proof. Its proof may be found elsewhere {Klein and Kaliski, 1979.}

> Let f be any unimodal map. Let x, y in [0,1] be given, x < y. Then, in the above defined Gray code order, rsk (x) <= lsk (y), for all k, and rs (x) <= ls (y).

III. A "Roadmap" for the Technical Developments That Follow

To aid the reader in the developments to come let us sketch out a "roadmap" of the remainder of the paper. Section IV examines certain recursions for calculating the exploded k-signature repertoires of unimodals and subbells. The role of the left signature of the peak value q in these recursion formulas is central. In Section V we examine certain necessary and sufficient conditions for the existence of fixed points having given finite or infinite signatures. This is followed in Section VI by an examination of the "orders" of fixed points and of the presence of chaotic regimes in unimodal and subbell functions. Necessary background material is introduced as needed, and the reader is often referred to the existing literature for proofs of many of the basic results.

IV. Recursions for Determining Signature Repertoires

We begin by discussing two recursions for obtaining the realizable k-bit signatures of a given unimodal function. By "realizable" signature we mean a signature of a regular point or the left or right signature of an irregular point, i.e. a signature in the exploded repertoires of f. The first is drawn from the cited references. Recall that the exploded k-signature repertoire is denoted by Sk.

Theorem 1: {Klein and Kaliski, 1979} For all k >=1,

S k+1 = {(0 . Sk) ^ [ls k+1(0), ls k+1(p)]}

U {(1 . Sk) ^ [rs k+1(p), rs k+1(1)]}

where . denotes concatenation, ^ denotes intersection, and the square brackets represent closed intervals in the < ordering on the (k+1)-bit sequences.

Since the k-bit signatures of the points 0 and 1 under a subbell function are respectively, 0**k and 10**k-1, the recursion is greatly simplified when f is a subbell. This simplification leads to an alternate method of determining realizable signatures for subbell mappings:

Theorem 2: Suppose that f is a subbell. Then for k>=1 :

(i) S1 = {0,1} and,

(ii) S k+1 = 0. S'k U 1. S'k

where S'k consists of those strings s in Sk for which s is <= lsk(q).

Proof: We need to cite the following Lemma without proof:

Lemma 1: When f is a subbell the exploded repertoires Sk are given by:

S1 = {0,1}

S k+1 = 0 . {Sk ^ [0**k, lsk(q)]}

U 1 . {Sk ^ [0**k, lsk(q)]}

(The proof of Theorem 2 resumes) Clearly, S1 is as given. As for the expression for S k+1, observe that Sk ^ [0**k , lsk(q)] = S'k. The expression for S k+1 then follows immediately from the Lemma. QED

The last theorem underscores the significance of the left signature of the peak q.In "signature space" a subbell becomes a one-parameter family of sequences determined completely by the left signature of the peak value.

V. On The Existence of Fixed Points of fk pq Having Given Finite and Infinite Signatures

In this section we are concerned with the existence of fixed points of fk pq for various values of k, having given finite and infinite signatures.These points are points x0 for which fk pq(x0) = x0 and their characterization is essential if one is to obtain a theory of the orbital behavior of the function f in question.

Fixed points eventually map into themselves. We are thus motivated to consider rotations of finite signatures. Our discussion begins here.

V,1 Let s be a k-bit sequence . Let $s^{\#}j$ be the sequence obtained by rotating s circularly j bits to the left, $0<=j< k$. We call $s^{\#}j$ the jth left rotation of s. We view $s^{\#}0$ to be equal to s. We say that $s^{\#}j$ is a rotation maximal of s, denoted rm(s), if $s^{\#}i <= s^{\#}j$ for i = 0,1,...,k-1. Note that the existence of a rotation maximal of any finite binary sequence is guaranteed since <= is a linear ordering of finite binary sequences of equal length. The value of j may not be unique however (for example in the string s= 011011).

V,2 Let us now examine sets which are such that any two points in it have the same k-signature, for some fixed k, under a given unimodal mapping (If the points are irregular then they are in the set if either their left or right signatures is the signature in question) . From a system theoretic point of view the common k-signatures may be seen as defining a type of isomorphism. It turns out that such points form a continuum; i.e.,the sets are intervals. This stems from the "monotonicity of signatures" property cited earlier. In fact, the intervals are closed. {Klein and Kaliski, 1979}.

V,3 Let f be a given unimodal map. Also let s = b0 b1... be an arbitrary infinite binary sequence. The k-signature bin of s (k>=1), with respect to f, is the set Ak =

{x\lsk(x) and/or rsk(x) = b0 b1 ... b k-1}. The signature bin of s with respect to f is the set A = {x\ls(x) and/or rs(x) = s}. As stated in V,2 , above, the sets Ak, if non-empty, are closed intervals in [0,1]. It is also true that A is a closed interval, and in fact is equal to ^ Ak (taken over all values of k.) {Klein and Kaliski, 1979}.

V,4 Infinite Sequences and Fixed Points of fk pq:

When considering infinite binary sequences, the concepts of rotation and rotation maximality need to be replaced by their infinite sequence analogues, shift and shift maximality, respectively. Let s1 and s2 be infinite binary sequences. We call s2 the jth left shift of s1 , denoted Lj(s1) , if s2 is obtained from s1 by deleting the leftmost j bits. Let S be the set of all left shifts (or tails) of the infinite binary sequence s. Then, the shift maximal of s is sup(S), with respect to the < order. We denote the shift maximal of s by sm(s). Note that, in general, sm(s) is not an element of S.

We now turn to three key questions that arise in exploring the issue of fixed point existence.

QUESTION 1: When is an infinite binary sequence s in the exploded signature repertoire S of a given subbell mapping f pq?

Theorem 3: Let s be an infinite binary sequence. Then s is in the exploded signature repertoire S of a subbell fpq if and only if

$$Lj(s) \quad <= \quad ls(q), \text{ for } j>=1.$$

Proof (->): Suppose s is in S, and is the left and/or right signature of x in [0,1]. Then every shift of s is in S , and is the left and/or right signature of a point in the orbit of x. Such points are, from the form of f pq, at most q. If q is not in the orbit of x then, from the monotonicity of signatures principle, Lj(s) <= ls(q) for all j>=1, and we are done. If q is in the orbit of x it is easy to argue that all shifts of s corresponding to iterates of x equal to q are equal to ls(q); all others, by monotonicity of signatures, are less than or equal to ls(q). The conclusion follows.

(<-): To prove the converse, we shall need the following:

Lemma 3: Let s be a given infinite binary sequence. Then s is in the exploded signature repertoire S of a unimodal f pq if and only if for all k every k-truncation of s is in

Sk. (By the k-truncation of s we mean its first k-bits.)

Proof of Lemma 3: The necessity of the condition is obvious. Sufficiency requires some discussion. Suppose every k- truncation of s is in Sk. Let {Ak} ,k=1,2,... be the k-signature bins based on s. Thus the Ak are nested and closed, and non- empty by hypothesis. We have noted that A = ^ Ak. Thus A is non-empty also, i.e. s is in S. QED.

(The proof of Theorem 3 resumes) We exploit the fact that f pq is a subbell by invoking Theorem 2. Suppose Lj(s) <= ls(q) for j = 1,2,... Let s = b0 b1 ... Since b0 is either 0 or 1, b0 is in S1. Similarly b1 is in S1. And as b1 is the first bit of L1(s) and L1(s) <= ls(q),by hypothesis,it follows that b1 <= ls1(q). It is immediate that b0 b1 is in S2, by Theorem 2. Now consider b0 b1 b2. It is easy to prove that b1 b2 is in S2 by an argument similar to that given above; further b1 b2 represent the first two bits of L1(s). Therefore , as L1(s) <= ls(q), b1 b2 <= ls2(q).Thus it is readily shown that b0 b1 b2 is in S3, again by Theorem 2 . By repeated use of the arguments above we can show that b0 b1 ...b k-1,the k-truncation of s, is a realizable finite signature of f pq for all k. Hence, by Lemma 3, s is in S. QED

We turn next to:

QUESTION 2: When is an infinite signature s in S the signature of a fixed point of a unimodal fk pq?

We have the following result:

Theorem 4: Let s = b0 b1 b2 ... be a given infinite binary sequence in S , for some unimodal f pq . Let k>=1 be given. If s is periodic with period k then there is a fixed point x of f pq of period k for which ls(x) and/or rs(x) = s.

(Note that the Theorem applies, in particular, to subbells, as well.)

Proof: We state the following lemma without proof.

Lemma 4: Let {Bi} , i=1,2,... be a collection of subsets of [0,1] and r:[0,1] -> [0,1] a given map. Then

r(^ Bi) [^ r(Bi)

(where the intersection is taken over all i)

(The proof of Theorem 4 resumes) Let {Ai} , i=1,2,... be the i-signature bins based on s. Then A = ^Ai is a nonempty closed interval as are the Ai , and A = {y\ls(y) and/or rs(y) = s}. Note that A1] A2] A3 ... and because of this nesting ^Amk =A also, m=1,2,...

Now, note that fk(Amk) [A (m-1)k, for all m >= 2, because of the periodicity of s. So ^fk(Amk) [^A (m-1)k, where the first intersection begins with m=1; the second with m=2. But the latter is (beginning with m=1) ^Amk = A,so we have, ^fk(Amk) [A. But by Lemma 4, fk(^Amk) [^fk(Amk) . Therefore fk(A) = fk(^Amk) [A and it is readily shown that f has a fixed point x of period k in A. Clearly this point has ls(x) and/or rs(x) = s. QED

In the case where the unimodal is, in fact, a subbell an even more powerful statement can be made.

Theorem 5: Let k>=1 and finite, and s an infinite binary sequence and fpq , a subbell, be given. Then there is a fixed point of fk pq with left or right signature s if

(i) sm(s) <= ls(q), and
(ii) s is periodic with period n such that divides k.

Proof: The proof we give is based on the fact that every periodic infinite binary sequence contains its shift maximal. Suppose s is periodic with period n which divides k, and sm(s) <= ls(q). As sm(s) is contained in s, Lj(s) <= ls(q), for j=1, 2,... Hence by Theorem 3, s is in the signature repertoire S of fpq , and the result follows immediately from Theorem 4. QED

V,5 Finite Sequences and Fixed Points of fk pq.

Before posing "QUESTION 3" we need to address several issues. Let us begin with the following finite sequence analogue of Theorem 3:

Theorem 6: Let f be a subbell function with peak value q. Let s be a k-bit sequence such that rm(s) <= lsk(q). Then every rotation of s is in Sk.

Proof: In order to prove the theorem, we shall need the following result, stated without proof: {Kwankam, 1979}

Lemma 5: Let f be a subbell function with break point p. Let s be a binary sequence of length k, such that for j = 0,...,k-1

(i) s*j <= lsk(p), if s*j begins with 0,

(ii) s*j >= rsk(p), if s*j begins with 1

Then s*j is in Sk for j = 0,1,2,...k-1

(The proof of Theorem 6 resumes.) Let f be a subbell function with breakpoint p and peak value q. We show by a reductio-ad-absurdum method that if s is as defined in the Theorem, every rotation s1, of s, satisfies the following conditions:

s1 <= lsk(p), if s1 begins with 0

and

s1 >= rsk(p), if s1 begins with 1.

We will then invoke Lemma 5 to complete the proof.

Assume, then, that there is some rotation s1, of s for which either s1= 0 b1...b k-1 > lsk(p) or s1= 1 b1...b k-1 < rsk(p).As p maps into q, we know that lsk(p) = 0 ls k-1(q) and rsk(p) = 1 ls k-1(q) Thus in the first case 0 b1 ... b k-1 > 0 ls k-1(q) and so b1 ...b k-1 > ls k-1(q). Note that if z1 and z2 are binary sequences of equal length with z1 > z2, then for any binary sequences z3 and z4 of equal length, z1 z3 > z2 z4. Thus b1 ...b k-1 0 = s1*1 > lsk(q). (In the above notation z1 = b1 ... b k-1, z2= ls k-1(q), z3=0, and z4= the kth bit of ls(q).) In the second case we have that 1 b1 ... b k-1 < 1 ls k-1(q) and so b1... b k-1 > ls k-1(q). Therefore, by the note b1 ... b k-1 1 = s1*1 > lsk(q).

We thus have the result that for either case s1*1 >

lsk(q), which cannot be since s1**1 <= rm(s1) <= lsk(q). Therefore the original assumptions must be false, i.e. every rotation s1 of s must be such that s1 <= lsk(p) if it begins with a 0; otherwise it must be >= rsk(p). By Lemma 5, then, every rotation of s is a realizable k-signature of f. QED

Remark 1: Let s be a binary sequence of finite length. Then rm(s**n) = rm(s)**n.

We shall need the following lemma as well in the developments below.

Lemma 6: Let x0 be a fixed point of a unimodal fk pq. Then either ls(x0) and/or rs(x0) is periodic of period k. Let s denote the first k bits of this periodic signature. Then there exists x1 in the orbit of x0 whose left k-signature is equal to rm(s). Furthermore, rm(s)**2 <= ls2k(q), and, when it is equal to ls2k(q), then ls(q) is, in fact, periodic, and equal to rm(s) rm(s) ...

Proof: Clearly sig(x0) is periodic with period k. If sig(x0) is regular then s is equal to sigk(x0) and it is equally apparent that the rotation maximal of s occurs as a subsequence of sig(x0). If this subsequence begins at position j in sig(x0), j>=0, then rm(s) is the k-signature of x1 = fj pq(x0). Furthermore sig(x1) = rm(s) rm(s) ... Note that when sig(x0) is regular p and hence q cannot occur in the orbit of x0, and thus all points in the orbit of x0 are less than q. It is immediate, from the monotonicity of signatures principle, that rm(s)**2 < = ls2k(q). If rm(s)**2 is < ls2k(q) there is nothing left to prove. Assume then that rm(s)**2 is equal to ls2k(q).

For ease of notation, write rm(s) as s1. Thus sig(x1) = s1 s1 .... Again, from the montonicity of signatures principle, sig(x1) <= ls(q). Assume that it is less than ls(q). We know by hypothesis that its first 2k bits agree with those of ls(q). It must be, then , that ls(q) is of the form s1**n z, where n>=2 and where the infinite sequence z does not begin with s1 and obeys s1**n z > s1 s1 ... . Suppose that sig(x1) and ls(q) differ for the first time at the nk+j th bit, with 0<j<k. Let m=nk, if n is even, and (n-1)k if n is odd. Consider L'm(s1 s1 ...) and L'm(ls(q)). The former is again s1 s1 ..., whereas the latter is z if n is even, and s1 z if n is odd. Note that in both cases (n even, n odd) an even number of s1's were deleted, and hence s1 s1 ... < L'm(ls(q)). Since these two sequences differ within the first 2k bits (since z does not begin with s1) it must be that the first 2k bits of L'm(ls(q)) are greater than s1 s1 = ls2k(q). So L'm(ls(q)) > ls(q). This is not possible by Theorem 3, for ls(q) is in S. The conclusion then is that ls(q) is equal to sig(x1) and is thus periodic and equal to rm(s) rm(s) ....

When x0 is irregular we argue as follows. p and q must occur in the orbit of x0, and no point greater than q is in its orbit. It is easy to demonstrate that ls(q) will be periodic of period k and that either ls(x0) or rs(x0) will be also, depending on the parity of the initial portion of sig(x0) before the first "-" occurs. (After this "-" the remainder of both ls(x0) and rs(x0) is ls(q).) So s is the initial k-bits of this periodic signature and rm(s) is lsk(q), by the monotonicity of signatures principle. Furthermore the point x1 having rm(s) as its left k-signature is q itself, and thus ls(x1) is equal to ls(q). With ls(q) periodic it is immediate that rm(s)**2 is equal to ls2k(q) and ls(q) = rm(s) rm(s) ... QED

We are now in a position to address the final question of section V:

QUESTION 3: When is a given k-bit sequence s the k-bit left and/or right signature of a fixed point of fk pq?

Let us refer to the k-signature s of Lemma 6 (i.e. the initial k bits of ls(x0) and/or rs(x0), whichever is periodic) as the "periodic" k-signature of x0.

Theorem 7: Let s be a given k-bit binary sequence and fpq a subbell function. Then there is a fixed point of fk pq with "periodic" k-signature s if and only if rm(s) **2 <= ls2k(q), with ls(q) = rm(s) rm(s) ... if rm(s) **2 = ls2k(q).

Proof: (->) This is immediate from Lemma 6.

(<-) Conversely, write rm(s) as z1 to simplify the discussion below. Consider the sequence z2 = s **n, for any n > =2. From Remark 1, rm(z2) = z1 **n. Now, by hypothesis, z1**2 <= ls2k(q), with ls(q) = z1 z1 ... in the case of equality. Thus, rm(z2) <= lsnk(q). By Theorem 6, then, z2 is a realizable finite signature of fpq . Thus every truncation of z2 is also a realizable finite signature of fpq. Note that this is true for all n > 1. Therefore by Lemma 3, the infinite sequence s s ..., made up of repetitions of s is a realizable signature of fpq. As s s ... is periodic with period k (or possibly a factor of k), Theorem 4 implies that there is a fixed point x, of fk pq with ls(x) and/or rs(x) = s s s .... s is clearly its "periodic" k-signature. QED

Signature-Distinct Unimodals and Subbells:

A unimodal F= fpq is said to be signature-distinct if sig(x) = sig(y) implies that x = y. A wide-class of signature distinct unimodals have been shown to exist

{Klein and Kaliski, 1979} and include all those unimodals which are "piecewise strictly expansive", i.e. for which there exists E > 1 such that for all x,y in [0,p], |F(x)-F(y)| >= E |x-y|, and similarly for all x,y in [p,1]. A consequence of signature-distinctness is that if x is not equal to y then no instance of sig(x) is equal to an instance of sig(y), either. {Klein and Kaliski, 1979}

When attention is restricted to the signature-distinct subbells a more powerful form of Theorem 7 can be proven:

Theorem 8: Let s be a given k-bit binary sequence and f pq a signature-distinct subbell function. Then there is a fixed point of fk pq with "periodic" k-signature s if and only if rm(s)**2 <= ls2k(q) with the further proviso,in the case of equality, that q be a fixed point of fk pq.

Proof:

(->) If such a fixed point exists then by Theorem 7 rm(s)**2 < = ls2k(q). In the case of equality ls(q) is periodic with period k and equal to rm(s) rm(s) ... So, by Theorem 4 there exists a fixed point x of fk pq whose left and/or right signature is equal to ls(q). Since f pq is signature-distinct x must, in fact, be equal to q.

(<-) If rm(s)**2 is < ls2k(q) the result is immediate from

Theorem 7. If equality holds and q is a fixed point of fk pq, it is easy to see that ls(q) must be periodic with period k. Since ls(q) begins with rm(s) rm(s) it must, in fact, be the case that ls(q) = rm(s) rm(s) ... The result is immediate from Theorem 7 then too. QED

VI . On Fixed Points and Their Orders

Our attention turns in this final section to understanding the nature of the fixed point regimes of unimodals and subbell functions. To simplify the notation in this section we omit the "pq" from fpq and fk pq, when convenient to do so.

Let us begin our development by discussing the notion of the order of a fixed point. First let us define the order of a sequence.

A finite sequence s is of order k if there exists k such that s can be written as s1 s1 ...s1, where s1 is a k bit sequence ,and where s1 cannot be similarly decomposed. Thus a sequence of only ones (or zeros) is of order one. The five bit sequence 10110 is of order five, whereas the six bit sequence 010101 is of order two and the eight bit sequence 10111011 is of order four.

x0 is a fixed point of order k of the mapping f if,

(i) fk (x0) = x0

and

(ii) fj (x0) <> x0, for j = 1,2,...,k-1

i.e., a fixed point of order k first maps into itself after k iterations under the given mapping. Note that if x is a fixed point of fk for some k and if the "periodic" k-signature of x is of order k then x is a fixed point of order k.

Certainly every unimodal map f with peak value greater than its break point has a regular fixed point with one-bit signature 1. This is clearly a fixed point of order one which we call the nontrivial fixed point of order one . Thus the k-signature of such a point is 1**k, and is in Sk. We are naturally motivated to ask if a k-bit sequence containing but a single zero might be too. Our next result shows that 101**k-2 is, indeed, a realizable signature provided that f obeys the restrictions below.

Theorem 9: Let f be a unimodal map with peak value q and break point p such that 0 and 1 are regular and such that

(i) ls2(q) = 10

(ii) sig2(0) = 00

and (iii) sig3(1) = 100

Then 101*k-2 is a realizable k-signature of f, for k >= 2.

Proof: Let f, p, and q be as stated. Clearly the Theorem is true for k=2. Assume then that k > 2. With ls2(q) = 10, then q>p, ls(p) = 010... and rs(p) = 110... Now 1**k-2 is a realizable signature, by our remarks above. Therefore, since sig k-1(0) < 01**k-2 <= ls k-1(p) we have that 01**k-2 is too, by Theorem 1. Next consider 101**k-2. Since rsk(p) < 101**k-2 < sigk(1), by the same recursion 101**k-2 is a realizable signature of f. QED

A subell satisfies all three conditions of the theorem provided ls2(q) = 10. Therefore,for such subbells,101**k-2 is always a realizable k-signature. It is also easily verified that this sequence is of order k.

Let us restrict our attention to these subbells in the remainder of this section, referring to them as "well-structured" subbells.

Theorem 10: Let f be a "well-structured" subbell. Then f has fixed points of order k, k odd, and >= 3, if and only if it has a fixed point, x0, of fk, whose "periodic" k-signature is equal to =101**k-2.

Proof: The proof of this theorem hinges on three properties of the sequence 101**k-2. First, as already noted,it is of order k. Secondly, it is its rotation maximal. The third property is stated as a lemma, also given without proof. {Kwankam, 1979}

Lemma 7: Let s be a k-bit sequence other than the trivial sequences 0**k and 1**k. If k is odd, then

rm(s) >= 101**k-2

(The proof of Theorem 10 resumes:)

(<-) By our earlier remarks concerning orders of sequences and orders of fixed points, the result is immediate, since 101**k-2 is of order k.

(->) Conversely, suppose f has fixed points of order k. Let y0 be one such fixed point. Assume that its "periodic" k-signature s is not equal to 101**k-2. (If it is, we are done). Then by Lemma 7, rm(s) > 101**k-2. Now, by Theorem 7, rm(s)**2 is less than or equal to ls2k(q). Thus, it is immediate that 101**k-2 101**k-2 < ls2k(q). Using Theorem 7 again, the result follows. QED

We have seen, in the foregoing, conditions governing the existence of fixed points of odd orders for "well-structured" subbells. We shall now show how the existence of fixed points of order k, implies the existence of fixed points of order k + 1, k + 2 , and k - 1, as long as k is odd and greater than one. Formally, we have:

Lemma 8: Let f be a "well-structured" subbell. If f has a fixed point of order k, k odd and k > 1, then f has fixed points of orders k + 1, k + 2, and k - 1.

Proof: Let k, an odd integer greater than one be given. Assume f has a fixed point of order k. Then by Theorem 10, there is a fixed point x0, of fk whose "periodic" k-signature is 101**k-2. From Theorem 7, it follows that

101**k-2 101**k-2 <= ls2k(q)

Consider the sequence 101**k-1 which we denote by s1. It is of order k + 1. Moreover, it is its rotation maximal. If we can show that s1 s1< ls 2k+2(q) , then by Theorem 7, there is a fixed point of f k+1 with "periodic" (k+1) -signature s1. Examine the leftmost k + 2 bits of s1 s1 and of (101**k-2)(101**k-2). With k odd, 101**k-1 1 < 101**k-2 10.Thus the leftmost 2k bits of s1 s1 are < (101**k-2)(101**k-2) and hence < ls2k(q). So s1 s1 itself is < ls 2k+2(q).

So, by Theorem 7, as mentioned, there is a fixed point of fk whose "periodic" k-signature is 101**k-1. Since this is of order k+1 it is immediate that the fixed point is also.

Consider next the sequence 101**k ,which we denote by s2. It is of order k+2, and is its own rotation maximal also. By considering the leftmost k+2 bits of s2 s2 and those of 101**k-2 101**k-2 and by arguing as we have done for s1, it can be readily shown that s2 s2 < ls 2k+4(q). Hence f has fixed points of order k + 2.

Finally consider the sequence 101**k-3 which we denote by s3. It is of order k - 1, and it is rotation maximal. This time, we examine the leftmost k + 1 bits of s3 s3 and of (101**k-2)(101**k-2) By arguments similar to those presented above for s1 and s2, it can be shown that s3 s3 < ls 2k-2(q). Therefore, f has fixed points of order k - 1. This completes the proof. QED

Therefore when a "well-structured" subbell has fixed points of odd order k, k > 1, it also has fixed points of orders k - 1, k + 1, and k + 2. Note that if k is odd, so is k + 2. The existence of fixed points of order k + 2 thus implies the existence of fixed points of orders k + 3 and k + 4 . k + 4 is also odd and so the argument can be repeated, ad infinitum. This leads, naturally, to the following result:

Theorem 11: Let f be as above. Then if f has a fixed point of order k0, with k0 > 1 and k0 odd, then f has fixed points of every order k >= (k0 - 1).

We conclude, from this theorem, that a "well-structured" subbell with fixed points of order three has fixed points of all orders. (Every subbell has a fixed point of order *one*.) This conclusion is very much like that of Li and Yorke {Li and Yorke, 1975}. If, we restrict our attention to signature distinct "well-structured" subbells, we can propose a method of determining the existence of fixed points of all orders which is considerably easier than the method implied by Li and Yorke's theorem.

Theorem 12: Let f be a signature-distinct "well-structured" subbell. Then f has fixed points of all orders if and only if rs3(q)=100.

Proof:(<-) The sufficiency of the condition is readily proved. Suppose rs3(q) is 100. If sig3(q) has no dash in it, then ls3(q) is also 100. Therefore as 101 101 < ls6(q), we have by Theorem 7, that f3 has a fixed point with three-bit "periodic" signature 101. This is clearly a fixed point of order three, which by Theorem 11 means f has fixed points of all order greater than one. As every subbell has a fixed point of order one, f has fixed points of all orders.

If sig3(q) has a dash in it, then sig3(q) must be 10- or else ls2(q) <> 10, a necessity for "well-structured" subbells. Therefore q is a fixed point of f3 . Furthermore, q is a fixed point of order 3. Thus f has fixed points of all orders.

(->) To prove that rs3(q) = 100 is a necessary condition, it is sufficient to show that if it is not met, there is at least one order of fixed points which is not part of the " fixed point repertoire" of f; such an order is three. Since 100 is the largest three bit sequence, it must be that rs3(q) < 100. Since ls2(q)=10, it follows that sig2(q) is 10 and, rs3(q) and ls3(q) must both be 101. Thus sig3(q) does not contain a dash. Therefore q is not a fixed point of f3. Now, from Theorem 3, ls(q) is shift maximal. So with ls3(q) = 101, it is easy to prove that ls6(q) <= 101101.

We can thus conclude that there is no fixed point of f3 with "periodic" signature 101, for were there to be one, then, from Theorem 6 (since f is signature distinct) it would have to be the case that ls6(q) = 101101, and that q is one of them, contradicting the above remarks. It is immediate that f has no fixed points of order three from Theorem 10. This completes the proof. QED

We therefore need examine only the right signature of the peak value of a "well-structured" signature distinct subbell to determine whether or not the subbell has fixed points of all orders. What is more important --from considerations of ecomomy of computation-- is that we need look at only its first three bits i.e., we need to compare with p the values of q, f(q) and f2(q)!

The family of "symmetric tent" maps shown in Figure 3 will exhibit this "chaotic" behavior (fixed points of all orders) provided that a>= (1+sqrt(5))/4, a fact that is readily demonstrated by showing that for such values of a, and only for such values, will rs3(q)= 100.

VII. Conclusions

This paper has presented a variety of results concerning the fixed point structure of certain maps on the unit interval. The underlying common factor of the developed theory has been that of the signature of a point and of its role in characterizing the map's orbital behavior. From an expositional point-of-view the signature-based theory is appealing due to its minimal dependence on advanced measure-theoretic concepts typically found in the current literature.

References

Kaliski, M. and Klein, Q. "Behavior of a Class of Nonlinear Discrete Time Systems", submitted to Mathematical Systems Theory, 1982

Klein, Q. and Kaliski, M. "Functional Equivalence in a Class of Autonomous One-Dimensional Nonlinear Discrete-Time Systems", Information and Control,42,2,1979

Collett,P. and Eckmann,J. "Iterated Maps on the Interval as Dynamical Systems", Birkhauser, 1980.

Li,T. and Yorke,J. "Period Three Implies Chaos", American Mathematical Monthly, 82, 1975.

Parry, W. "Symbolic Dynamics and Transformations of the Unit Interval", Trans. American Mathematical Society,122, 1966.

Ballieul,J., Brockett,R. and Washburn, R. "Chaotic Motion in Nonlinear Feedback Systems", IEEE Trans. on Circuits and Systems, CAS-27,11,1980.

Feigenbaum,M. "Universal Behavior in Nonlinear Systems", Los Alamos Science, 1980.

May, R. and Oster,G. "Bifurcations and Dynamic Complexity in Simple Ecological Models", American Naturalist, 1976.

Kwankam, S. "A Structure Theory for the Fixed Points of the Iterates of Certain One-Dimensional Nonlinear Functions Defined Over the Unit Interval", Ph.D. Dissertation, Northeastern University Dep't of Electrical Engineering,Boston, MA, 1979.

AD A135 257 ASYNCHRONOUS DISCRETE CONTROL OF CONTINUOUS PROCESSES 2/2
(U) NORTHEASTERN UNIV BOSTON MA M E KALISKI ET AL.
30 JUL 83 AFOSR-TR-83-0877 F49620-82-C-0080
UNCLASSIFIED F/G 9/3 NL

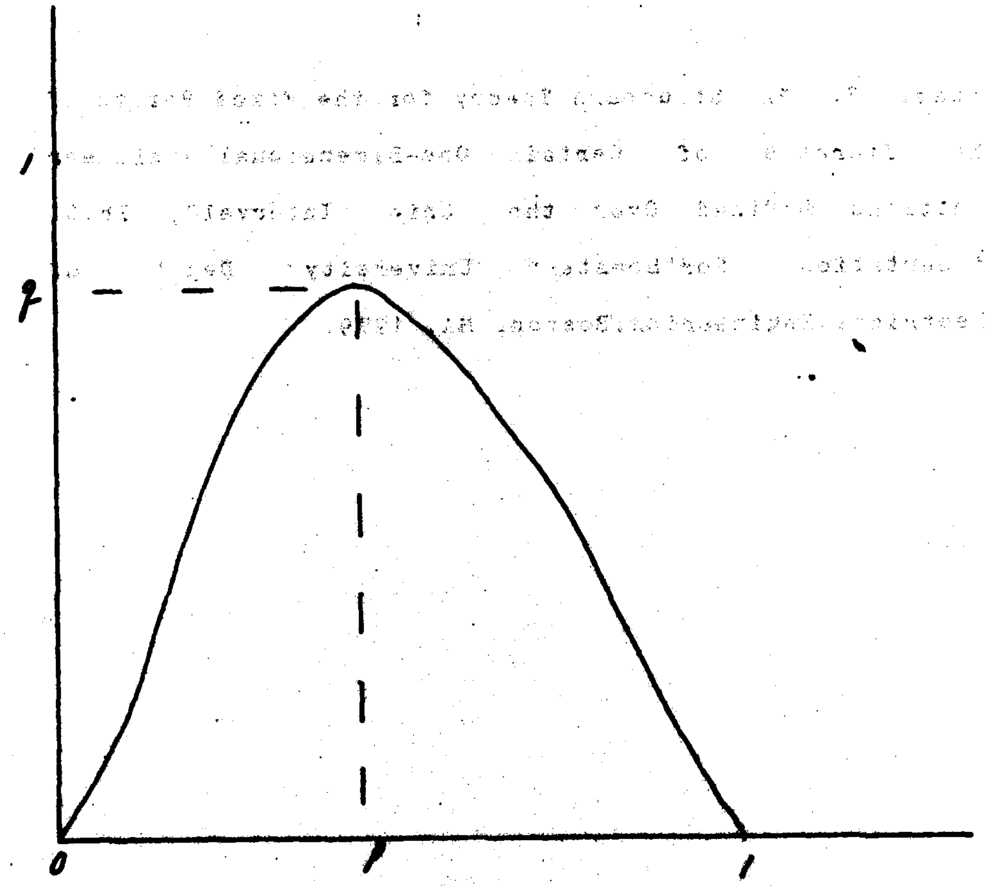


Figure 1

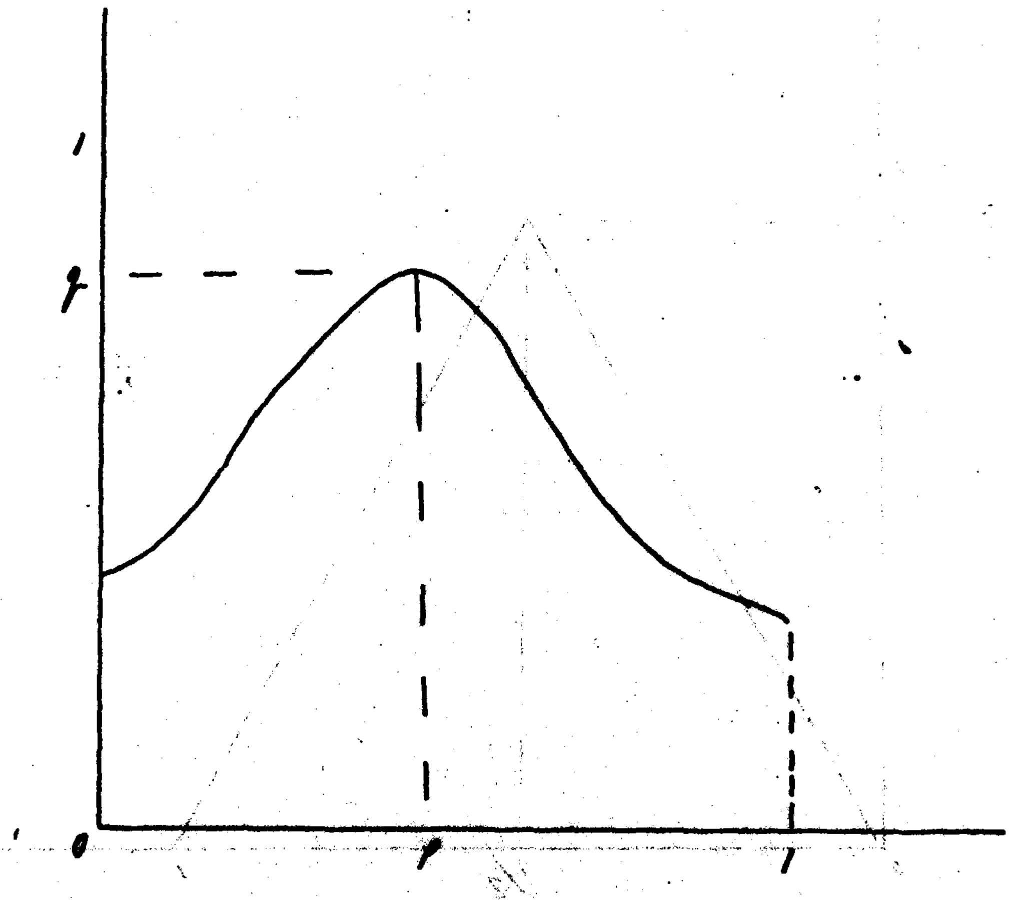


Figure 2

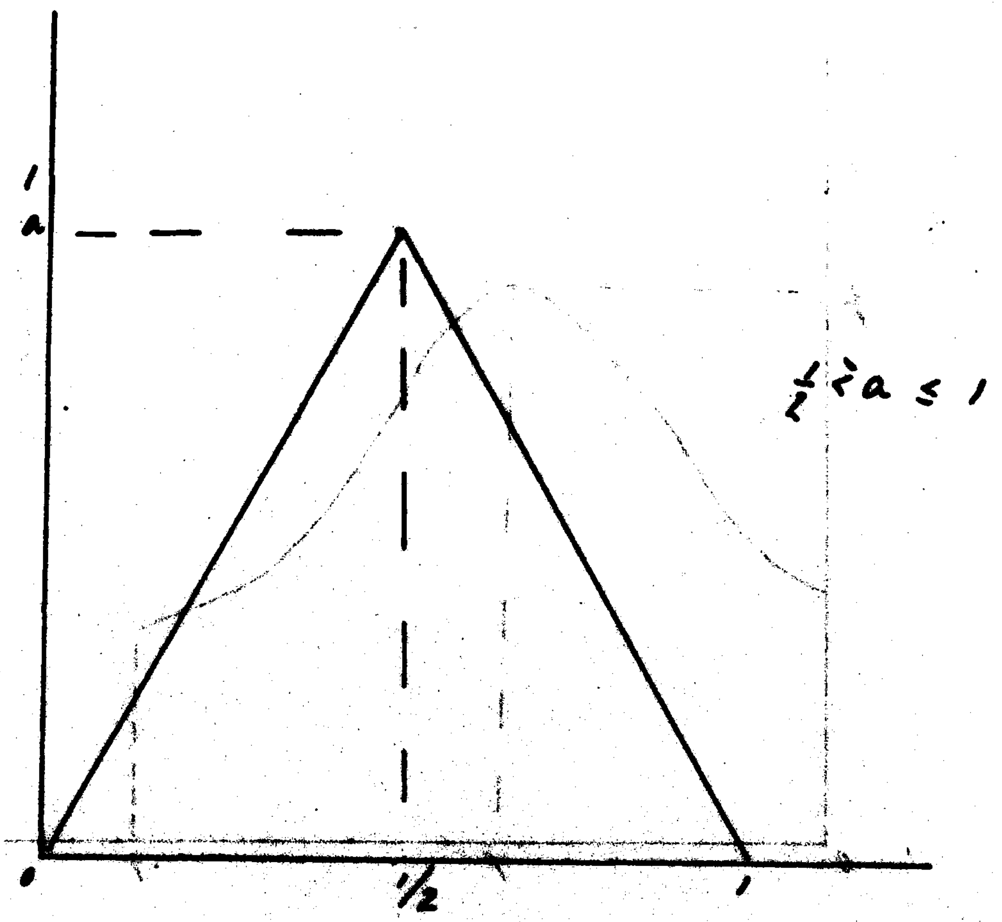


Figure 5